Fakultät für Elektrotechnik und Informationstechnik
Technische Universität München

Modeling and Analyzing Dynamics of Visual Processes With Representation Learning

Alexander Sagel

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzende(r): Prof. Dr. Wolfgang Utschick

Prüfer der Dissertation:

1. Priv.-Doz. Dr. habil. Hao Shen
2. Prof. Dr.-Ing. Klaus Diepold

Die Dissertation wurde am 16.06.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 04.02.2022 angenommen.

Alexander Sagel. *Modeling and Analyzing Dynamics of Visual Processes With Representation Learning.* Dissertation, Technische Universität München, Munich, Germany, 2022.

Chair of Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de/`.

Acknowledgment

I would like to take the opportunity to express my gratitude to all the people that have made this thesis possible.

First and foremost, I thank my thesis supervisor Prof. Dr.-Ing. Klaus Diepold for the opportunity and freedom to carry out my research at LDV, and the valuable advice during this time. Furthermore, I am particularly thankful to PD Dr. Hao Shen and PD Dr. Martin Kleinsteuber for all the things that I have learned about research, mathematics and work ethic from them.

My time as a doctoral candidate would not have been as productive and fun, if it was not for all the people I had the opportunity to work with. Most of all, I consider myself lucky to have crossed paths with the great individuals at GOL, LDV, and fortiss, including Matthias, Julian, Martin K., Peter, Clemens, Dominik, Wei, Martin G., Mingpan, Stefan, Zhiwei and Tianming. Thank you for the collaborations, discussions, proofreading, and the coffee breaks. In this regard, I should also not let go unmentioned the wonderful people of the SIROCCO team at inria for making me feel welcome during my year in Rennes, and rendering my time there both enjoyable and scientifically fruitful.

On a more personal note, I will forever be thankful to my parents for always being understanding and encouraging. I could not have wished for a better moral support.

Finally, and most importantly, I want to say *merci* to Anne, for two reasons. First of all, she gave me the right idea at a critical moment that has prompted me to investigate deep generative models for video. But more fundamentally, I am thankful for making these last years for me simply magical, despite all the hard work and the emotional drain it entailed.

Abstract

This thesis discusses the topic of visual processes, i.e., cyclic or repetitive visual phenomena that can be interpreted as multi-dimensional stochastic processes. Such phenomena appear in different technical and natural settings. They play an important role in visual intelligent systems, which raises the question of how to accurately describe their statistical properties. It turns out that state space models in which the states follow a linear, multi-dimensional autoregressive transition dynamic have considerable appeal in modeling visual processes. This appeal is reflected in mathematical simplicity, computational tractability and versatility of applications. In this thesis, different state space models are proposed in which linear state transitions are combined with non-linear observation functions to describe high-dimensional visual processes.

First, a model created by interpreting the visual process videos as sequences of histograms is discussed. The state space model is learned using kernel PCA and a distance measure is proposed that can be applied to classify dynamic textures and dynamic scenes. Additionally, a variation of this model is proposed that uses Scattering transforms for computing the histograms and achieves state-of-the-art classification performance on a dynamic texture dataset.

Afterwards, a variational autoencoder is employed to jointly learn a neural network based observation function and the linear state transition model for generative modeling of visual processes. As an application, video synthesis is considered. The resulting synthesis algorithm is simple in both the mathematical sense as well as in terms of computational burden, once the network has been trained. Nevertheless, it can reproduce sequences of considerable visual complexity.

To enhance the synthesis results, the subject of post-processing is discussed. This is done by investigating the problem of increasing the resolution given a synthesized video. To this end, a super-resolution mechanism is introduced that, unlike many contemporary algorithms, does not require learning on external data. The results compete well with other learning-free algorithms on common 2D image datasets. On visual processes, the algorithm yields first proof-of-concept results that pave the way for further research.

Overall, this thesis demonstrates that visual processes can be accurately and easily described by state space models with linear latent dynamics, and proposes ways to employ such models in a variety of real-world applications using techniques from representation learning.

Contents

List of Publications

This thesis is based on results from the following peer-reviewed publications.

A. Sagel, D. Meyer and H. Shen
Texture Retrieval Using Scattering Coefficients and Probability Product Kernels
International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA), pp. 506-512, Aug. 2015.

A. Sagel and M. Kleinsteuber
Alignment Distances on Systems of Bags
IEEE Transactions on Circuits and Systems for Video Technology, 28(10), pp. 2551-2561, Oct. 2018.

A. Sagel and H. Shen
Dynamic Variational Autoencoders for Visual Process Modeling
IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3677-3681, May 2020.

A. Sagel, A. Roumy and C. Guillemot
Sub-Dip: Optimization on a Subspace With Deep Image Prior Regularization and Application to Superresolution
IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2513-2517, May 2020.

A. Sagel, J. Wörmann and H. Shen
Dynamic Texture Recognition via Nuclear Distances on Kernelized Scattering Histogram Spaces
IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3755-3759, Jun. 2021.

List of Figures

List of Tables

List of Symbols

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\mathbf{1}_n$	Vector in $\mathbb{R}^n$ containing ones as its entries, i.e. $[1, \dots, 1]^\top$
diag$(a_1, \dots, a_n)$	A square, diagonal matrix with diagonal entries given by $a_1, \dots, a_n$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Indexing

$(\boldsymbol{a})_i$	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
$(\boldsymbol{A})_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{U}_{(n)}$	Matrix containing the n first columns of $\boldsymbol{U}$
$\boldsymbol{\Sigma}_{(n)}^{(n)}$	Matrix containing the n first rows and columns of $\boldsymbol{\Sigma}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudoinverse of $\boldsymbol{A}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$
$\lVert\boldsymbol{a}\rVert_p$	ℓ_p norm of vector $\boldsymbol{a}$
$\lVert\boldsymbol{a}\rVert$	ℓ_2 norm of vector $\boldsymbol{a}$
$\lVert\boldsymbol{A}\rVert_F$	Frobenius norm of matrix $\boldsymbol{A}$
$\lVert\boldsymbol{A}\rVert_2$	Spectral norm of matrix $\boldsymbol{A}$
$\lVert\boldsymbol{A}\rVert_*$	Nuclear norm of matrix $\boldsymbol{A}$
$\mathrm{tr}(\boldsymbol{A})$	Trace of matrix $\boldsymbol{A}$

Sets

$\mathbb{A}, \mathcal{A}$	A set
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integers
$\mathbb{N}$	The set of natural numbers
$\{0, 1\}$	The set containing 0 and 1
$\{1, \ldots, n\}$	The set of all integers between 1 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathrm{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$	The vector space spanned by $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
L^p	The Lebesgue space of p-integrable functions
$\mathrm{GL}(n)$	The general linear group of $n \times n$-dimensional, real, invertible matrices
$\mathrm{O}(n)$	The orthogonal group of $n \times n$-dimensional, real matrices
$\mathrm{SO}(n)$	The special orthogonal group of $n \times n$-dimensional, real matrices
$\mathrm{St}(n, d)$	The Stiefel manifold of matrices in $\mathbb{R}^{d \times n}$ with orthonormal columns

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral w.r.t. $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$p(\mathrm{a})$	A probability distribution over a variable a
$\mathrm{a} \sim p$	Random variable a has distribution p
$\mathbb{E}_{\mathrm{x}\sim p}[\mathbf{x}]$	Expectation of $\mathbf{x}$ w.r.t. p
$\mathrm{Cov}(\mathbf{x})$	Covariance of $\mathbf{x}$, $(\mathrm{Cov}(\mathbf{x}))_{i,j} = \mathrm{Cov}((\mathbf{x})_i, (\mathbf{x})_j)$
$\mathrm{Cov}(\mathbf{x}, \mathbf{y})$	Covariance of $\mathbf{x}$ and $\mathbf{y}$, $(\mathrm{Cov}(\mathbf{x}, \mathbf{y}))_{i,j} = \mathrm{Cov}((\mathbf{x})_i, (\mathbf{y})_j)$
$D_{\mathrm{KL}}(p\|q)$	Kullback-Leibler divergence of p and q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g

List of Acronyms

BoS	Bag of Systems
BoW	Bag of Words
CNN	Convolutional neural network
DIP	Deep image prior
DVAE	Dynamic variational autoencoder
FE	Feature extraction
FID	Fréchet inception distance
GAN	Generative adversarial net
k-NN	k-nearest neighbors
KDE	Kernel density estimator
KLDS	Kernelized linear dynamic system
LBP	Local binary patterns
LDS	Linear dynamic system
Max SV	Maximum singular value
(N)SHKS	(Normalized) Scattering Histogram Kernel Subspace
PCA	Principal component analysis
PSNR	Peak signal-to-noise ratio
RGB	Red/green/blue
RNN	Recurrent neural net
SoB	System of Bags
SVD	Singular value decomposition
TV	Total variation
VAE	Variational autoencoder
VAR	Vector autoregressive

1. Introduction

Computer vision and machine learning have experienced considerable growth in interest as research fields during the recent years. While progress on a large variety of data modalities has been made, these fields owe much of their success to accurately describing, modeling and processing still images. However, for many technical applications this is not sufficient. Still images are instantaneous snapshots of the real world. As such, they sometimes do not provide profound insight into the real-world phenomenon that has caused them. The world that surrounds us is constantly in motion. When we look around us, there is rarely ever a moment in which everything stands still. For technology that relies on building an accurate model of real-world events, this has important implications.

For instance, let us imagine that we want to build a system that aims at detecting abnormal crowd behavior, e.g. mass panics, from real-time surveillance footage. Such happenings are difficult to capture from still images, but are usually distinguishable if we put the pictures into a temporal context. As another example, we can think of a video retrieval system, in which we try to trawl all videos in a database based on certain movements or dynamic patterns. Again, we cannot expect to capture the essential information just by looking at isolated video frames. Instead, we need a way to account for what is happening as the frames evolve temporally. Many of the visual phenomena that we encounter every day are quite repetitive and we tend to not take note of them, considering them as “background noise” to our everyday lives. This background noise is the subject of this thesis. More specifically, it looks into the problem of finding an appropriate dynamic model to describe *visual processes*.

In the scope of this thesis, let us define *visual process* as a *stochastic process in which every observation is a video frame*. Furthermore, let us assume that every video frame is described by a vector in $\mathbb{R}^d$. This vector can directly contain the pixel values of the frame or some other feature representation thereof. A visual process is thus a sequence of random variables defined over a probability space. Visual processes, by their nature, are not just characterized by their randomness but also by some degree of predictability. Depending on the application, it might thus be sensible to make additional assumptions about the statistical properties of the process, such as ergodicity or stationarity.

For the sake of understanding, let us consider a few examples. Stable physical systems tend to oscillate around equilibrium points. This is true for a simple spring pendulum, as depicted in Figure 1.1, as it is for complex industrial machinery. Such

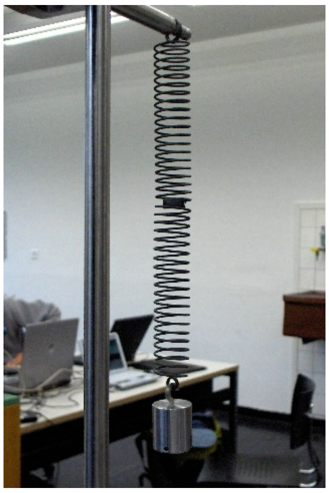


Figure 1.1.: Motion captures of harmonic oscillations exhibited by spring pendulums can be considered an example of visual processes [45].

systems are often equipped with different types of sensors in order to monitor the operation and detect failures before they occur. The ubiquity and low cost of cameras allows for the possibility to take such measurements visually. Motion captures of stable mechanical systems are an example of visual processes. In industrial applications, this kind of data is of interest in fault detection scenarios [29]. Since visual data is easy and cheap to obtain, visual process models can aid in detecting when the system is behaving in an anomalous way. Similar motion patterns also occur in footage of human or animal movements [19] and are of interest in applications such as human-computer interaction.

Another example of visual processes are *dynamic textures* [38], also known as *temporal textures* [117]. We can define them as temporally evolving image textures, or alternatively, as visual motion patterns with some form of stationarity or repetitiveness in both time and space. They constitute the most classical and, arguably, well-studied type of visual processes. Typical dynamic textures are natural phenomena such as ocean waves or vegetation moving in the wind. However, dynamic textures appear in virtually every area of our daily lives in the form of waving flags, chimney smoke or campfire flames. Dynamic textures are employed in a variety of applications. In the past, dynamic texture models have been applied in feature based video recognition [87], video segmentation [39] and motion detection [35], for instance.

Dynamic scenes are one further important example of visual processes. In the same manner as the term “dynamic texture” denotes a moving textural pattern, “dynamic scene” refers to a moving real-world scene. Typically, a dynamic scene con-

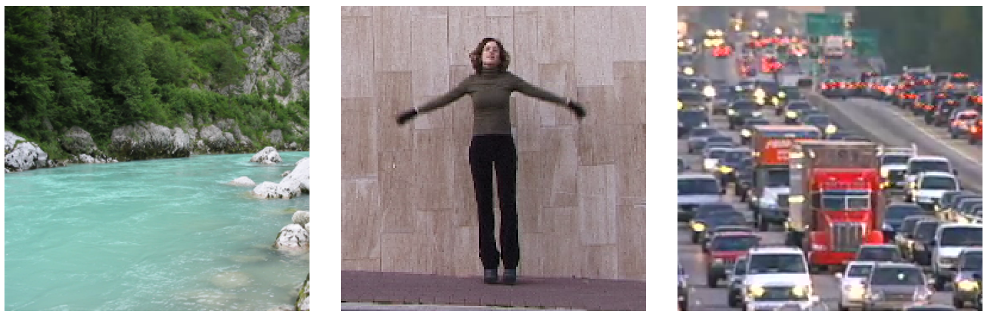


Figure 1.2.: Examples of visual processes. a) Flowing water b) Person doing jumping jacks [19] c) Highway [34]

sists of several objects moving in a regular and predictable manner, such as in traffic scenes or sport events. Similarly, wide-angle captures of industrial sites, such as wind parks or assembly lines, can be also considered dynamic scenes. Figure 1.2 depicts examples of typical visual processes.

1.1. Remarks on Notation

The notation in this thesis is based on the book [46] for which LaTeX code is publicly available[1]. A detailed notation table can be found in the front matter of this thesis. It is worth noting that random variables, e.g. $\mathbf{A}, \mathbf{x}, \mathrm{c}$, are written using an upright font, as opposed to regular matrix-valued, vector-valued or scalar terms, e.g. $\boldsymbol{A}, \boldsymbol{x}, c$, that are written in italic. Furthermore, sets are generally written either using blackboard bold, e.g. $\mathbb{M}$, or calligraphic typeface, e.g. $\mathcal{M}$. Blackboard bold letters are typically used for sets containing data samples or feature representations thereof. Calligraphic letters usually indicate abstract concepts, such as function spaces or sets of parameter tuples. Unless stated otherwise, the term *SVD* refers to the full singular value decomposition with the singular values arranged in a descending order.

1.2. Typical Problems in Visual Processes

The different types of visual processes, ranging from dynamic textures to video surveillance footage are an important field of research in computer vision. To help putting the content of this thesis into a larger context, it is sensible to bring oneself

[1] `https://github.com/goodfeli/dlbook_notation`

to mind the considerably large application range of this research field. Common problems in visual processes are listed in the following.

- *Classification* is a classical task in supervised learning and as such is also one of the oldest problems in visual process research [105]. As we will see later on, the stationary character of visual processes facilitates classification. The reason is that it permits us to build upon methods employed in classical still-image classification. This is of course not true if the task is to classify according visual process properties that cannot be inferred from individual frames. Compared to still-images, it is much harder to gather a sufficient amount of training data, which poses a considerable difficulty for classification.

- Another important application is *clustering* [2]. Unlike the supervised learning problem of classification, assigning data points to *clusters* of semantically similar data samples is a typical example of *unsupervised learning*. In addition to grouping different videos according to their visual appearance, clustering has also been used to characterize the flow of vehicles in traffic videos [27].

- *Retrieval* [11] denotes the task of identifying a set of videos in a database that match a search query. When the search query consists of keywords that are then used to trawl the meta-data of annotated video sequences, we refer to the problem as *meta-data based* retrieval. Another situation occurs in *content based* retrieval. In that case, the query consists of a video sequence that is first processed by a feature extraction algorithm to extract a signature from it. This signature is then compared to signatures in the database, using an appropriate similarity measure. The retrieval then returns the sequences most similar to the query in the database. Classification and retrieval tasks are summarized by the term *recognition* in this thesis.

- *Frame prediction* refers to the task of forecasting one or several video frames, given a sequence of frames that has preceded them [83]. The capability to accurately predict the future of a visual process is of interest both in semantic problems, e.g. predicting human behavior [22] and in pixel-level tasks, e.g. compression or restoration of a video [78].

- A purely generative problem is visual process *synthesis* [38] that encompasses generating natural-looking sequences that follow the same stochastic properties as a real-world visual process. Synthesis should not be confused with prediction, since the former is stochastic and the latter is deterministic in nature. This means in particular that synthesis cannot be evaluated using common image distances. While the most obvious applications of visual process synthesis are in the creative field, such as video game design or animation of paintings via *style transfer* [119], more technical applications, such as compression [30], hole filling [76], or data augmentation for supervised learning are also imaginable.

- *Anomaly detection* refers to the task of identifying patterns in a visual process that contradict some notion of normality. While a typical application is in industrial settings, anomaly detection goes beyond detecting failures in technical systems. For instance, identifying unexpected occurrences in road traffic or crowd dynamics is a possible domain of application [79].

- Spatial and temporal *segmentation* of visual processes has also attracted some interest [39]. Often, segmentation requires some form of clustering as an intermediate step. Possible applications are, for instance, preprocessing of medical or satellite data.

The aforementioned and similar problems related to visual processes often require an appropriate way to describe them. Whether or not a visual process descriptor is "appropriate" might depend on which one of the vast number of applications it is supposed to be deployed in. Roughly speaking, we can identify two types of visual process descriptors.

Observation based approaches refer to all methods that aim to extract features from a video with the property of capturing distinguishing characteristics of the visual process. Examples include spatio-temporal frequency transforms [41], features based on optical flow [122] or features computed via convolutional deep neural networks [96]. Typically, these kinds of descriptors are used in discriminative scenarios such as classification or segmentation of dynamic textures.

Model based approaches follow another path. They aim at uncovering the hidden driving forces of a visual process. Such methods are generative in the sense that they describe sequences by means of a model that could have possibly *generated* them. Model based approaches are in general more difficult to design and cannot be expected to compete as well on supervised learning problems as observation based techniques trained for a specific task. However, they are generally less task-specific and thus more versatile, since the model is typically not learned with one specific application in mind. Particularly, tasks that require the knowledge of statistical properties of a visual process, such as synthesis or anomaly detection, can benefit from a model-based approach. These advantages are the reason this thesis revolves exclusively around model-based methods to describe visual processes.

Before proceeding to specifying the research question and outlining how model-based approaches can be used to solve different computer vision problems related to visual processes, it is necessary to narrow down and characterize the type of models the discussion in this thesis concentrates on.

1.3. Linear Dynamic Systems

Discrete, dynamic processes can be described by means of recurrence equations, i.e., equations that indicate how an observation at a certain time relates to earlier observations. The underlying assumption roots in our idea about the world that is based on causality. In other words, we usually assume that any observation we make is a result of processes that have occurred at an earlier time. When processed digitally, visual phenomena are observed at discrete points $t \in \mathbb{Z}$ in time. We make the assumption here that for any visual process, we can find a *recurrence function* ϕ that, at any time t, computes the current observation $\boldsymbol{y}_t$ from previous observations $\boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \dots$, and an input vector $\boldsymbol{u}_t$. This yields the recurrence equation

$$\boldsymbol{y}_t = \phi(\boldsymbol{u}_t, \boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \dots). \tag{1.1}$$

The variable $\boldsymbol{u}_t$ stands for actions applied at time t that are not a direct consequence of previously occurred situations. It models the non-deterministic aspect of the visual process. For instance, when we observe outdoor scenes under stormy conditions, we can predict how certain objects like vehicles behave in the near future given the current and recent observations, while others move in a chaotic or unpredictable fashion, e.g. autumn leaves on the ground. In that case, $\boldsymbol{u}_t$ captures the movement trajectories of such objects.

To describe natural visual processes, the observations $\boldsymbol{y}_t$ with $t \in \mathbb{Z}$ must be of a considerably high dimension in order to account for the data demand posed by the resolution of natural images. This means that inferring a model, i.e., finding a recurrence function ϕ, requires regressing in a very high-dimensional space. This is bound to fail due to the curse of dimensionality [15], i.e., the high number of parameters to be estimated given a limited data corpus.

On the other hand, assuming that the video frames depict an actual physical phenomenon, it is fair to imagine it as a mechanism that is driven by only a small number n of latent variables. In that case, each observation $\boldsymbol{y}_t \in \mathbb{R}^d$ has a low-dimensional representation $\boldsymbol{x}_t \in \mathbb{R}^n$ and there is an *observation function*

$$\Gamma : \mathbb{R}^n \to \mathbb{R}^d \tag{1.2}$$

that maps each *latent state* $\boldsymbol{x}_t$ to the respective observation $\boldsymbol{y}_t$. The intrinsic dynamic of the latent states is governed by a function φ that computes the state at $t+1$ from the m preceding states and the input vector $\boldsymbol{u}_t$. We call m the *order* of the system. These definitions yield the model equations

$$\boldsymbol{x}_t = \varphi(\boldsymbol{u}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \dots, \boldsymbol{x}_{t-m}) \tag{1.3a}$$
$$\boldsymbol{y}_t = \Gamma(\boldsymbol{x}_t). \tag{1.3b}$$

The function φ operates on a *latent* space with a lower dimension than the domain of ϕ and is thus also easier to learn from data. We refer to models described by Eq. (1.3) as *state space models* [101].

A simple and popular choice for the latent state process in Eq. (1.3a) is a *vector autoregressive* model (VAR). It models the recurrence function φ as a linear combination using a set of matrices $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_{m-1}$ and an input vector that is assumed to be a sample $\boldsymbol{v}_t \in \mathbb{R}^d$ of i.i.d.[2] zero-mean Gaussian noise. This yields the formulation

$$\boldsymbol{x}_{t+1} = \sum_{i=0}^{m-1} \boldsymbol{A}_i \boldsymbol{x}_{t-i} + \boldsymbol{v}_t. \tag{1.4}$$

The dynamic texture model as proposed in [38] has popularized *linear dynamic systems* (LDS) in the modeling of visual processes. According to the work [38], an LDS is a state space model in which the latent state transitions are described by a first-order VAR process and the observation function Γ is affine, i.e., described by an *observation matrix* $\boldsymbol{C} \in \mathbb{R}^{d \times n}$, and a constant *offset vector* $\boldsymbol{y} \in \mathbb{R}^d$. Denoting by $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ a transformation matrix that accounts for the noise covariance, the model in Eq. (1.3) becomes

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{v}_t, \tag{1.5a}$$

$$\boldsymbol{y}_t = \bar{\boldsymbol{y}} + \boldsymbol{C}\boldsymbol{x}_t, \tag{1.5b}$$

where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is called the *state transition matrix*, and the input term $\boldsymbol{v}_t$ is modeled as i.i.d. standard[3] Gaussian noise that is statistically independent of $\boldsymbol{x}_t$. To summarize, the term *LDS* refers to a combination of a latent first-order VAR process and an affine observation model. Without loss of generality, we can assume that $\boldsymbol{C}$ has full rank. If that is not the case, we can find an equivalent model with a latent space of a lower dimension $\tilde{n} = \operatorname{rank}(\boldsymbol{C}) < n$, and an observation matrix $\tilde{\boldsymbol{C}} \in \mathbb{R}^{d \times \tilde{n}}$, where the column spaces of $\boldsymbol{C}$ and $\tilde{\boldsymbol{C}}$ are identical.

1.4. Semi-linear Dynamic Systems

The model described in Eq. (1.5) has important advantages as discussed in the following.

1. It is of a mathematically very simple form. Eq. (1.5a) is essentially a linear transformation perturbed by zero-mean i.i.d. Gaussian noise. Both of these operations are well understood and studied, which cannot be said of non-linear sequential models such as architectures based on *recurrent neural nets* (RNN) [116].

2. This facilitates system analysis. For instance, the authors of [53] argue in favor of LDS's because they simplify system identification. Furthermore, it

[2]Independent and identically distributed
[3]Zero mean and unit variance

suffices to take a look at the spectrum of state transition matrix $\boldsymbol{A}$, if we want to investigate the long term behavior of an LDS. To name an example, one important property that is often expected from a sequential model is stability. In informal terms, a model described by Eq. (1.5a) is *stable*, if we can guarantee it not to amplify the input noise beyond any bounds. This property can be guaranteed to hold if all singular values of $\boldsymbol{A}$ are smaller than 1, because then, every input $\boldsymbol{B}\boldsymbol{v}_t$ contributed by the noise term fades out over time. On the other hand, many applications of the model are mathematically and numerically easy to carry out. Contemporary models for video synthesis, for example, often require a tedious optimization procedure for every sequence to be synthesized [119, 127], while LDS sequences can be easily sampled by generating standard Gaussian noise and feeding it to the model.

3. Another advantage of systems described by Eq. (1.5) is their versatility. Models acquired by supervised end-to-end learning are often constrained to perform a limited number of tasks. By contrast, an LDS not only provides a model for the statistics of a visual process, but can also be applied as a feature for recognition tasks, making it more capable of multi-task learning [24] scenarios.

These observations are important arguments that support LDS's as a design choice for visual processes models. Beyond that, finding and employing LDS-like models becomes increasingly important as LDS's reemerge as a subject of interest in machine learning. Recent work in the field has proposed LDS's as interpretations of stochastic gradient descent [50] or as a baseline for reinforcement learning algorithms [84]. In [64], this increase of interest has been also explained by numerical and analytical simplicity. Furthermore, it is believed [64] that LDS's can approximate RNNs for sequential modeling, but without carrying their burden of mathematical intractability. Exploiting this capability would require methods to learn, analyze and apply such LDS based models.

That being said, the LDS model in Eq. (1.5) has an obvious drawback. Realistically speaking, real-world visual phenomena are not linear and randomness in visual processes cannot be entirely captured by Gaussian noise. So while Eq. (1.5) does have considerable appeal in terms of the mathematical formulation, we cannot expect it to be a sufficiently complex model of the real world. However, there is a way to enhance the model that significantly increases its capability to describe visual phenomena, while still maintaining an effectual level of simplicity. Note that the advantages regarding the simplicity of LDS models discussed in this chapter are mainly due to Eq. (1.5a) since it is the part containing recurrence. On the other hand, Eq. (1.5b) limits visibly the capabilities of the model since it constrains the observations produced by the visual process to be contained in an affine subspace of $\mathbb{R}^d$. Thus, replacing this mapping by some other, possibly non-linear function $\Gamma : \mathbb{R}^n \to \mathbb{M}$ with $\mathbb{M} \subset \mathbb{R}^d$ can greatly enrich the model capacity without losing much of its simplicity and explainability. This yields the model equations

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{v}_t, \tag{1.6a}$$

$$\boldsymbol{y}_t = \Gamma(\boldsymbol{x}_z). \tag{1.6b}$$

Eq. (1.6) will remain the guiding theme of this thesis.

1.5. Research Problem Formulation

This thesis follows the presumption that we are given a sequence

$$\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N \in \mathbb{R}^d \tag{1.7}$$

of observed video frames generated by a visual process. Based on this sequence, the aim is to infer a dynamic model that can plausibly explain the succession of frames. Such a model gives us insight into the mathematical laws of a visual process and provides a framework for solving the numerous applications discussed previously. In the following, it will be assumed that the visual process in question can be described by Eq. (1.6) with sufficient accuracy. Since every visual process is described by a different state transition $\boldsymbol{A}$, noise covariance transformation $\boldsymbol{B}$ and observation function Γ, describing a visual process via Eq. (1.6) requires determining these parameters.

Based on these requirements, the main research question of this thesis can be formulated as follows.

> Given an observed video sequence generated by some visual process, how can we infer the parameters $\boldsymbol{A}, \boldsymbol{B}$ and Γ, such that Eq. (1.6) provides an accurate dynamic model of the underlying visual process?

It should be noted that accuracy in this context depends on the intended application. Certain applications require the model to capture the behavior of the visual process up to pixel level, so that one can accurately predict object movements, for instance. Other applications might only need structural information to be retained, as is the case in certain methods for dynamic texture segmentation, to name an example.

Inferring the semi-linear system parameters of a visual process using a history of observations poses difficulties at different levels. The observation function Γ needs to be chosen such that its range is an accurate model of the set containing the observations of the visual process. Beyond that, video sequences offer only few samples to train on, when seen in relation to the number of pixels in its individual video frames. Learning the observation function Γ from data is thus prone to overfitting.

In machine learning, the problem of finding “approriate” parameterizations of data is studied within the field of *representation learning* [16]. Practically, the above research question entails the problem of finding a representation learning method that is capable of parameterizing observed sequences of real-world visual processes as trajectories of VAR noise. Importantly, representation learning techniques that are known to be successful for still-image samples might not work on visual processes. Given some learned function Γ, it may be possible that for any observation of a visual process, one can find a data point in the domain of Γ which is mapped to the respective observation, but the latent points in the domain of Γ do not follow any temporal structure. This has also implications on the choice of the VAR parameters $\boldsymbol{A}$ and $\boldsymbol{B}$. The two matrices can be learned separately from Γ, which is computationally simpler, but may not accurately capture the dynamic behavior of the visual process. Learning the VAR parameters and the observation function jointly generally leads to a more reliable estimate of the spatio-temporal properties, but requires more intricate learning algorithms.

It is important to note that the approaches in this thesis are not the first to employ the model in Eq. (1.6). For instance, two algorithms proposed in this thesis are based on the kernel-based method introduced in [26] that also uses this formulation.

Once we have determined an algorithm to infer $\boldsymbol{A}, \boldsymbol{B}$ and Γ from an observed sequence of a visual process, we can employ them to solve practical problems related to visual processes. However, finding the right dynamic model is often insufficient to solve particular tasks. While the model provides statistical laws that can explain the visual process, further measures need to be usually taken in order to put the inferred model into action. This is the case, for example, in the problem of designing a video retrieval framework for visual processes. Besides computing the model, the problem also requires deciding on the proper representation of the video frames from which the model is inferred and on a concise descriptor of the dynamic model that can be stored in a database. Furthermore, a similarity measure that matches query descriptors to models in the database needs to be defined.

Such complementary problems are inherently related to model inference as they are needed for putting the learned model into practice. So in addition to the problem of statistical modeling, this thesis also explores the following complementary scientific challenges.

- *Feature extraction*: The exact content of the vectors in Eq. (1.7) depends on the application. Often, each observation at time t is present as a simple vectorization of the respective video frame. Other applications benefit from first transforming the video frames into a certain feature representation. Deep learning based methods for inferring the observation function Γ often do not require such a transformation, because it is learned implicitly, e.g. within the encoder of an auto-encoding architecture. For less versatile representation learning approaches, e.g. kernel-based methods, a hand-crafted representation can aid in capturing the essential application-specific information from each video frame. In this thesis, several known and novel feature representations based on distributions of visual characteristics are discussed and methods for dynamic model inference are presented for the case when the visual process video is described as a sequence of such features.

- *Metric learning*: Many common recognition tasks require a notion of similarity for the entities to be recognized. If the task consists of classifying, clustering or retrieving visual processes based on dynamic model parameters, metric learning needs to take into account the specific mathematical structure exhibited by the set that these parameters occupy. The problem of metric learning is covered by considering previously introduced distance metrics on pairs of LDS's and discussing how they can be generalized to semi-linear dynamic systems in a way such that they retain the semantic properties essential to a particular application and visual process type.

- *Post-processing*: If the dynamic model is capable of synthesis, the visual quality of the synthesized sequence may be practically limited by accuracy of the learned model, as well as by hardware resources. One possibility to overcome this problem is by applying post-processing steps once the sequence is generated. Such steps can include techniques for image enhancement to increase the visual quality, but also style-transfer methods for artistic animation. This thesis investigates the capability to improve the resolution of synthesized visual processes using classical still-image superresolution techniques.

The two main applications investigated in this thesis are *classification* and *synthesis*. However, a guiding principle in designing the models presented in this thesis has been to make them as broadly applicable as possible. For instance, even though the similarity measures introduced in the following are only evaluated on classification problems, they can be theoretically also employed to carry out clustering or retrieval tasks. Likewise, the synthesis algorithm presented later on can be easily adapted for anomaly detection and prediction methods. This emphasis on *task-agnostic* models is the reason that none of the presented techniques relies on learning the model on external datasets, as is common in classical "deep" and "shallow" machine learning.

1.6. Thesis Outline and Key Contributions

This thesis is structured as follows. Following Chapter 1 that provides an introduction into visual processes and possible mathematical models for them, Chapter 2 reviews the LDS model in Eq. (1.5). It motivates and describes an algorithm based on *principal component analysis* (PCA) to compute the model parameters. After this introduction, it is demonstrated how to apply the model to video synthesis using the computed LDS parameters by sampling from autoregressive noise and transforming these noise samples by means of the observation matrix. Furthermore, common distance measures that are used in classification and clustering of LDS's are described in detail.

Afterwards, Chapter 2 provides an overview of common non-linear representation learning techniques that are either employed in this thesis or are closely related to it. This includes *Kernel PCA* [108] as well as more recent approaches from deep learning, and, in particular, deep generative models [66, 47] that transform low-dimensional Gaussian noise to samples of some learned data distribution.

The main scientific contribution of this thesis revolves around two realizations of the model described by Eq. (1.6). One realization is designed for classification of dynamic textures and dynamic scenes. To do so, the observations of Eq. (1.6b) are represented as histogram based feature vectors living in a low-dimensional subspace of a kernel feature space created by Kernel PCA. This approach resembles the human action recognition algorithm presented in [28] and is studied in Chapter 3. It works by transforming the visual process videos to sequences of feature vectors, where each one of the vectors is created by converting a video frame into a histogram describing the distribution of certain visual characteristics. This is motivated by the fact that such distribution based features have demonstrated success in other areas of computer vision. The resulting sequences of histograms are then used to construct dynamic models using an inference algorithm relying on histogram kernels previously introduced in literature. The classification is performed via k*-nearest neighbors* (k-NN) and *nearest class centers* (NCC), which poses several challenges to the framework. To begin with, these techniques require a measure of dissimilarity, but this measure needs to account for the fact that the same dynamic system can be described using different parameters of Eq. (1.6). The Alignment distance [3] is proven to handle this ambiguity quite well. However, the Alignment distance is defined for LDS's as described by Eq. (1.5). Therefore, it is generalized to handle kernel-based systems as defined in Eq. (1.6) and important properties of the distance are shown, including that it is a metric. Another challenge is that kernel-based video descriptors are data-hungry and grow with the video length. To overcome this problem, a *Nyström* interpolation [40] based approach is introduced that reduces the memory usage and computational burden.

Chapter 4 develops the technique presented in Chapter 3 further. For this purpose, a histogram feature based on the Scattering transform is proposed and a kernel adapted to these features is introduced. Thereafter, it is argued that the most distinctive property in dynamic texture recognition is captured in the appearance of a visual process, while the dynamics can be often neglected. This supposition allows for a simplification of the distance measure employed in Chapter 3 that greatly reduces computational effort while achieving state-of-the-art classification results on common dynamic texture datasets. Unlike Chapter 3, Chapter 4 is not evaluated on dynamic scenes, but focuses entirely on dynamic textures.

The second realization of Eq. (1.6) is discussed in Chapter 5. It introduces a generative model for visual processes using *variational autoencoders* [66]. This realization can be used, among others, for video synthesis. Since Variational autoencoders learn a function that maps Gaussian noise to some learned data distribution, a Variational autoencoder is employed to learn the observation function Γ in Eq. (1.6b). More precisely, system parameters $\boldsymbol{A}, \boldsymbol{B}$ and the observation function Γ are learned using a generative architecture that models the observation function via a convolutional neural net. Unlike many other approaches based on linear and semi-lnear dynamic systems, all the parameters are learned jointly, which is important to ensure that sequences of the visual process actually correspond to VAR trajectories in the latent state space. This is done by including an additional linear building block into the neural network, the so-called *dynamic layer*, that is learned as part of the neural architecture during the training process. Later, it is used to compute the parameters $\boldsymbol{A}$ and $\boldsymbol{B}$. For this to work, assumptions on the stationarity and Markov property of the visual process are made. These assumptions are crucial for the proposed adaptation of the classical Variational autoencoder that can learn the state transiton matrix and the process noise covariance as part of its architecture, along with the observation function. The presented framework is experimentally evaluated on a series of synthesis experiments.

Following the description of a synthesis algorithm based on variational autoencoders in Chapter 5, the problem of enhancing the synthesis quality is considered. To investigate the capability of increasing the synthesis resolution via post-processing, a super-resolution algorithm is introduced in Chapter 6. Based on the *deep image prior* [121], it leverages the advantages of deep convolutional neural networks, but does not require training on external image databases, unlike conventional deep learning algorithms. In the experimental section, it is demonstrated that the proposed method can outperform other learning-free algorithms for super-resolution and the applicability to post-processing of synthesized visual processes is investigated.

Lastly, Chapter 7 closes with a conclusion that summarizes the results of this thesis and discusses the relevance to current and prospective developments in research. It is followed by an appendix that provides additional theoretical discussion and experimental results.

2. Mathematical Preliminaries

The guiding research question in this thesis revolves around the problem of finding dynamic system models for visual processes. For linear models that follow the dynamics described by Eq. (1.5), theoretical analysis and methods for model inference and application are largely covered in literature [38, 39, 105, 130, 27]. The challenge in this thesis is to find *semi-linear* models as described by Eq. (1.6). In order to generalize the linear formulation to the semi-linear form, this chapter provides technical backgrounds on two important topics. First, the LDS model in Eq. (1.5) is discussed in-depth. Inference based on PCA is presented, and the problems of distance measurement and synthesis using the inferred model are introduced. This discussion lays the groundwork for understanding both the advantages and disadvantages of LDS's as visual process models, and walks through important concepts that will be generalized to non-linear interpretations later on in this thesis.

Afterwards, this chapter takes a look at different methods in data representation. The approaches will be later employed to describe Γ in Eq. (1.6b), thus making the generalization from linear to semi-linear models possible in the first place. To this end, it is required to find an appropriate model for the set of observations $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$ produced by a visual process. In the following, we refer to this set as *observation space* and denote it by $\mathbb{M} \subset \mathbb{R}^d$.

We assume that it is a compact subset of $\mathbb{R}^d$. Not every representation learning method is suitable for modeling the observation space of a visual process. For instance, classical manifold learning methods such as ISOMAP [118] or Locally Linear Embedding [102] may be a tempting choice, as they have a convincing geometrical motivation. On the downside, however, they do not result in an explicit or implicit representation of the observation function Γ. The aim of the second part of this chapter is to provide an overview over representation learning algorithms that can be exploited in the context of visual processes. In particular, we will consider kernel based representation learning and generative models in deep learning.

2.1. Linear Modeling

LDS models of visual processes are traditionally computed in two steps, with the first step constituting the inference of an appropriate subspace model for the video frames, and the second step being a mean squared error minimization problem to infer the latent VAR model [38]. One may assume that learning both aspects jointly

Algorithm 1 : PCA

```
  Input : Data matrix $\boldsymbol{Y} \in \mathbb{R}^{d \times N}$, target dimension $n \in \mathbb{N}$
1 $\bar{\boldsymbol{y}} \leftarrow \frac{1}{N}\boldsymbol{Y}\mathbf{1}_N$ ;                    // Computing data mean
2 $\boldsymbol{Y}^{\mathrm{c}} \leftarrow \boldsymbol{Y} - \bar{\boldsymbol{y}}\mathbf{1}_N^\top$ ;                           // Centering
3 $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V} \leftarrow \mathrm{SVD}(\boldsymbol{Y}^{\mathrm{c}})$;       // Singular Value Decomposition
  Output : Truncated left SVD matrix $\boldsymbol{U}_{(n)}$, data mean $\bar{\boldsymbol{y}}$
```

can improve the performance. However, we will see in this chapter, that for linear models, learning the VAR model and the subspace separately should not cause a great disadvantage. The model is thus discussed as it is introduced in [38] and usually used in computer vision literature, i.e., by learning the observation function and the latent state dynamics separately.

Principal Component Analysis

One of the most basic assumptions to make about data is that of linearity. Let us imagine that we are given a finite set of observations in $\mathbb{R}^d$. If the entire set is actually located on a low-dimensional affine subspace of $\mathbb{R}^d$, there is an invertible affine transformation that can map each data sample in the set to a lower-dimensional representation. In reality, the linear subspace assumption holds only approximately and any affine transformation that maps to a lower-dimensional space will lead to a loss of information. The idea of PCA is to find the affine transformation that minimizes this loss with regards to the ℓ_2 norm. It turns out that this transformation can be computed by the SVD of the data matrix, after subtracting the arithmetic average of the data samples from its columns.

Algorithm 1 summarizes the procedure of inferring the affine representation mapping from a set of data via PCA. As the input, it requires a data metrix $\boldsymbol{Y} \in \mathbb{R}^{d \times N}$ that contains the training samples $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ as its columns. First, the empirical mean $\bar{\boldsymbol{y}}$ of the observations is computed and subtracted from each observation, yielding the centered matrix $\boldsymbol{Y}^{\mathrm{c}}$. Let us write the SVD of $\boldsymbol{Y}^{\mathrm{c}}$ as

$$\boldsymbol{Y}^{\mathrm{c}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top. \tag{2.1}$$

The algorithm then returns the sub-matrix $\boldsymbol{U}_{(n)}$ containing the first n columns of $\boldsymbol{U}$, i.e. the leading n left singular vectors of $\boldsymbol{Y}^{\mathrm{c}}$, and the data mean $\bar{\boldsymbol{y}}$. Once the parameters $\boldsymbol{U}_{(n)}, \bar{\boldsymbol{y}}$ have been found, they can be used to represent a new data point $\boldsymbol{y}$ via

$$\boldsymbol{x} = \boldsymbol{U}_{(n)}^\top(\boldsymbol{y} - \bar{\boldsymbol{y}}). \tag{2.2}$$

Conversely, the reconstruction operation can be performed via

$$\tilde{\boldsymbol{y}} = \boldsymbol{U}_{(n)}\boldsymbol{x} + \bar{\boldsymbol{y}}. \tag{2.3}$$

Note that the representations $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ of the training set itself can be obtained directly from the right singular vectors of $\boldsymbol{Y}^{\mathrm{c}}$. Let

$$\boldsymbol{\Sigma}_{(n)}^{(n)} = \operatorname{diag}(\sigma_1, \sigma_2, \dots) \in \mathbb{R}^{n \times n} \tag{2.4}$$

denote the left upper submatrix of $\boldsymbol{\Sigma}$, where $\sigma_1, \sigma_2, \dots$ stand for the respective singular values. Then, we can write

$$\begin{aligned} &\begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_N \end{bmatrix} \\ =&\boldsymbol{U}_{(n)}^{\top} \begin{bmatrix} \boldsymbol{y}_1 - \bar{\boldsymbol{y}} & \cdots & \boldsymbol{y}_N - \bar{\boldsymbol{y}} \end{bmatrix} \\ =&\boldsymbol{U}_{(n)}^{\top} \boldsymbol{Y}^{\mathrm{c}} \\ =&\boldsymbol{\Sigma}_{(n)}^{(n)} \boldsymbol{V}_{(n)}^{\top}. \end{aligned} \tag{2.5}$$

PCA can be exploited for finding LDS models of visual processes, as will be discussed in the following.

Formulation as a Prediction Problem

Recall the LDS from Chapter 1 described by Eq. (1.5). Let the matrix

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_N \end{bmatrix} \in \mathbb{R}^{d \times N} \tag{2.6}$$

contain N temporally ordered observations of a visual process. In the classical scenario, the columns of $\boldsymbol{Y}$ are simply the vectorized RGB or grayscale video frames of a video sequence depicting the visual process. Given a latent state space dimension n, a key question is how to infer the parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ of Eq. (1.5) from $\boldsymbol{Y}$, such that they could be used to model the visual process.

In system and control theory, such problems are solved by *numerical algorithms for system identification* (N4SID) [91]. Here, we are presented with a slightly simpler version of this problem than the one typically described in control literature. The crucial difference is that Eq. (1.5) does not contain any control signals. The system is autonomous in the sense that it does not account for any external input except for the Gaussian process noise.

In order to infer the system parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ from $\boldsymbol{Y}$, we may be tempted to formulate a likelihood maximization problem, since this is a standard approach in machine learning. However, such a problem would not be well defined, as $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ probably do not lie directly in an n-dimensional subspace of $\mathbb{R}^d$. An alternative approach is to formulate the task as a prediction problem. Then, the parameters are determined in a way such that they can be used to estimate the best prediction $\boldsymbol{y}_{t+1}^{\mathrm{pred}}$ of the upcoming frame from the current frame $\boldsymbol{y}_t$. For this, only the parameters $\boldsymbol{A}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ describing the deterministic part of the process are needed. The following discussion serves to show that PCA provides a good approximation of the optimal model according to this motivation.

Note that if the parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ and the current observation $\boldsymbol{y}_t$ are known, and the Moore-Penrose pseudo-inverse of $\boldsymbol{C}$ is written as $\boldsymbol{C}^+$, the expectation-based prediction of $\boldsymbol{y}_{t+1}$ is given by

$$\begin{aligned} \boldsymbol{y}_{t+1}^{\text{pred}} &= \boldsymbol{C}\mathbb{E}_{\mathbf{v}_t \sim \mathcal{N}(0,\boldsymbol{I}_n)} \left[\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\mathbf{v}_t\right] + \bar{\boldsymbol{y}} \\ &= \boldsymbol{C}\boldsymbol{A}\boldsymbol{x}_t + \bar{\boldsymbol{y}} \\ &= \boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^+(\boldsymbol{y}_t - \bar{\boldsymbol{y}}) + \bar{\boldsymbol{y}}. \end{aligned} \tag{2.7}$$

By choosing the ℓ_2 norm as the error measure, we can then write the problem of estimating $\boldsymbol{A}$ and $\boldsymbol{C}$ from $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ as

$$\min_{\boldsymbol{A}\in\mathbb{R}^{n\times n}, \boldsymbol{C}\in\mathbb{R}^{d\times n}, \bar{\boldsymbol{y}}\in\mathbb{R}^d} \sum_{t=1}^{N-1} \left\|\boldsymbol{y}_{t+1} - \boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^+(\boldsymbol{y}_t - \bar{\boldsymbol{y}}) - \bar{\boldsymbol{y}}\right\|^2. \tag{2.8}$$

The arithmetic mean of a finite set of points in $\mathbb{R}^d$ minimizes the sum of squared distances to these points. As a consequence, the objective in Eq. (2.8) is minimized for $\bar{\boldsymbol{y}}$ under the condition

$$(\boldsymbol{I}_n - \boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^+)\bar{\boldsymbol{y}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \boldsymbol{y}_{t+1} - \boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^+\boldsymbol{y}_t. \tag{2.9}$$

This condition is *approximately* fulfilled for the choice

$$\hat{\boldsymbol{y}} = \frac{1}{N} \sum_{t=1}^{N} \boldsymbol{y}_t, \tag{2.10}$$

where the approximation relies on the assumption

$$\frac{1}{N-1} \sum_{t=2}^{N} \boldsymbol{y}_t \approx \frac{1}{N} \sum_{t=1}^{N} \boldsymbol{y}_t \approx \frac{1}{N-1} \sum_{t=1}^{N-1} \boldsymbol{y}_t, \tag{2.11}$$

which states that the empirical mean approximation is roughly the same, whether one uses all N observations, or only the first or last $N-1$ observations, to compute it.

Like in the previous subsection, let us identify

$$\boldsymbol{Y}^{\text{c}} = \boldsymbol{Y} - \hat{\boldsymbol{y}}\mathbf{1}_N^\top, \tag{2.12}$$

and for its columns, analogously,

$$\boldsymbol{y}_t^{\text{c}} = \boldsymbol{y}_t - \hat{\boldsymbol{y}}, \tag{2.13}$$

for all t. Substituted into Eq. (2.8) with $\bar{\boldsymbol{y}} = \hat{\boldsymbol{y}}$, this yields

$$\min_{\boldsymbol{A}\in\mathbb{R}^{n\times n},\boldsymbol{C}\in\mathbb{R}^{d\times n}} \sum_{t=1}^{N-1} \|\boldsymbol{y}^{\mathrm{c}}_{t+1} - \boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^{+}\boldsymbol{y}^{\mathrm{c}}_{t}\|^2. \tag{2.14}$$

We can assume that $\boldsymbol{C}$ is full-rank here, as argued in Section 1.3. Furthermore, we can also fix $\boldsymbol{C}$ to have orthonormal columns, i.e., to lie on the *Stiefel manifold* $\mathrm{St}(n,d)$. The reason is that if $\boldsymbol{C}$ does not have orthonormal columns, it can be written as a product of a matrix $\tilde{\boldsymbol{C}} \in \mathrm{St}(n,d)$ and a full-rank matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$. The summands in Eq. (2.14) then become

$$\left\|\boldsymbol{y}^{\mathrm{c}}_{t+1} - \tilde{\boldsymbol{C}}\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{-1}\tilde{\boldsymbol{C}}^{\top}\boldsymbol{y}^{\mathrm{c}}_{t}\right\|^2, \tag{2.15}$$

and $\boldsymbol{A}$ can be replaced by its similarity transformation $\tilde{\boldsymbol{A}} = \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{-1}$ in the optimization problem of Eq. (2.14). Therefore, Eq. (2.14) can be written as

$$\min_{\boldsymbol{A}\in\mathbb{R}^{n\times n},\boldsymbol{C}\in\mathrm{St}(n,d)} \sum_{t=1}^{N-1} \left\|\boldsymbol{y}^{\mathrm{c}}_{t+1} - \boldsymbol{C}\boldsymbol{A}\boldsymbol{C}^{\top}\boldsymbol{y}^{\mathrm{c}}_{t}\right\|^2. \tag{2.16}$$

Finally, let us denote by $\boldsymbol{C}_{\perp} \in \mathrm{St}(d-n,d)$, a Stiefel matrix with the property

$$\boldsymbol{C}_{\perp}^{\top}\boldsymbol{C} = 0. \tag{2.17}$$

By defining

$$\begin{aligned} \boldsymbol{Y}^{\mathrm{c}}_{-} &= \begin{bmatrix}\boldsymbol{y}^{\mathrm{c}}_1 & \boldsymbol{y}^{\mathrm{c}}_2 & \cdots & \boldsymbol{y}^{\mathrm{c}}_{N-1}\end{bmatrix} \in \mathbb{R}^{d\times N-1} \qquad \text{and} \\ \boldsymbol{Y}^{\mathrm{c}}_{+} &= \begin{bmatrix}\boldsymbol{y}^{\mathrm{c}}_2 & \cdots & \boldsymbol{y}^{\mathrm{c}}_{N-1} & \boldsymbol{y}^{\mathrm{c}}_{N}\end{bmatrix} \mathbb{R}^{d\times N-1}, \end{aligned} \tag{2.18}$$

we can rewrite the objective in Eq. (2.16) as

$$\begin{aligned} &\sum_{t=1}^{N-1} \left\|\boldsymbol{C}^{\top}\boldsymbol{y}^{\mathrm{c}}_{t+1} - \boldsymbol{A}\boldsymbol{C}^{\top}\boldsymbol{y}^{\mathrm{c}}_{t}\right\|^2 + \left\|\boldsymbol{C}_{\perp}^{\top}\boldsymbol{y}^{\mathrm{c}}_{t+1}\right\|^2 \\ =&\left\|\boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{+} - \boldsymbol{A}\boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{-}\right\|_F^2 + \left\|\boldsymbol{C}_{\perp}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{+}\right\|_F^2, \end{aligned} \tag{2.19}$$

where only the first term depends on $\boldsymbol{A}$. Thus, the optimal solution for $\boldsymbol{A}$ is determined using the pseudo-inverse as

$$\hat{\boldsymbol{A}} = \boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{+}(\boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{-})^{+}, \tag{2.20}$$

so that the problem in Eq. (2.16) boils down to

$$\min_{\boldsymbol{C}\in\mathrm{St}(n,d)} \|\boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{+}(\boldsymbol{I}_n - (\boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{-})^{+}\boldsymbol{C}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{-})\|_F^2 + \|\boldsymbol{C}_{\perp}^{\top}\boldsymbol{Y}^{\mathrm{c}}_{+}\|_F^2. \tag{2.21}$$

Algorithm 2 : LDS Modeling via PCA

```
  Input : Video Sequence $\boldsymbol{Y} \in \mathbb{R}^{d\times N}$, target dimension $n \in \mathbb{N}$
1 $\bar{\boldsymbol{y}} \leftarrow \frac{1}{N}\boldsymbol{Y}\mathbf{1}_N$ ;                    // Computing constant offset
2 $\boldsymbol{Y}^{\mathrm{c}} \leftarrow \boldsymbol{Y} - \bar{\boldsymbol{y}}\mathbf{1}_N^\top$ ;                                    // Centering
3 $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V} \leftarrow \mathrm{SVD}(\boldsymbol{Y}^{\mathrm{c}})$;       // Singular Value Decomposition
4 $\boldsymbol{C} \leftarrow \boldsymbol{U}_{(n)}$ ;                               // Observation matrix
5 $\begin{bmatrix}\boldsymbol{x}_1 & \cdots & \boldsymbol{x}_N\end{bmatrix} \leftarrow \boldsymbol{\Sigma}_{(n)}^{(n)}\boldsymbol{V}_{(n)}^\top$ ;               // Latent space states
6 $\boldsymbol{A} \leftarrow \arg\min_{\tilde{\boldsymbol{A}} \text{ s.t. } \|\tilde{\boldsymbol{A}}\|_2<1} \sum_{i=1}^{N-1} \|\boldsymbol{x}_{t+1} - \tilde{\boldsymbol{A}}\boldsymbol{x}_t\|_2^2$ ;          // State
   transition matrix
7 $\boldsymbol{B} \leftarrow \frac{1}{\sqrt{N-1}}(\sum_{i=1}^{N-1}(\boldsymbol{x}_{t+1} - \boldsymbol{A}\boldsymbol{x}_t)(\boldsymbol{x}_{t+1} - \boldsymbol{A}\boldsymbol{x}_t)^\top)^{\frac{1}{2}}$;          // Noise
   covariance transform
  Output : LDS parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$
```

We can minimize the second term $\|\boldsymbol{C}_\perp^\top \boldsymbol{Y}_+^{\mathrm{c}}\|_F^2$ by choosing $\boldsymbol{C}$ to be the matrix containing the leading n left singular vectors of $\boldsymbol{Y}_+^{\mathrm{c}}$, but that is generally not true for the first term. However, while this would generally not be the optimal solution for $\boldsymbol{C}$ taking into account both terms, it would still bring the first term close to 0, if $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ contain video frames from a real-world motion capture. This is due to how subspace projections affect natural images. Typically, projecting a video sequence on a low-dimensional principal subspace computed via PCA will create a blurred-out version of the video that no longer contains the fast-changing features of the original sequence and in which the succeeding video frames closely resemble each other. Hence, the approximation

$$\boldsymbol{C}^\top \boldsymbol{Y}_+^{\mathrm{c}} \approx \boldsymbol{C}^\top \boldsymbol{Y}_-^{\mathrm{c}} \tag{2.22}$$

holds, and therefore, the right-multiplication of $\boldsymbol{C}^\top \boldsymbol{Y}_+^{\mathrm{c}}$ by the projection matrix $(\boldsymbol{C}^\top \boldsymbol{Y}_-^{\mathrm{c}})^+ \boldsymbol{C}^\top \boldsymbol{Y}_-^{\mathrm{c}}$ in Eq. (2.21) has little impact. This justifies choosing $\boldsymbol{C}$ as the matrix that contains the first n singular vectors of the sequence.

Once the solution $\hat{\boldsymbol{C}}$ of Eq. (2.21) is determined, $\hat{\boldsymbol{A}}$ can be identified via Eq. (2.20), and the prediction error in the state space can be used to estimate the process noise model, i.e., $\hat{\boldsymbol{B}}$.

Theoretically, the preceding derivations suffice to construct an algorithm that estimates the LDS parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ from a sequence of observations. In reality, it is often necessary to adjust the procedure in a way such that the inferred system is stable. This can be done by constraining the spectrum of the state transition matrix to abide

$$\|\boldsymbol{A}\|_2 < 1. \tag{2.23}$$

Algorithm 2 summarizes the typically applied steps for LDS parameter estimation, while taking care of the requirement in Eq. (2.23).

Algorithm 3 : Synthesis via LDS

```
Input : System Parameters $\boldsymbol{A} \in \mathbb{R}^{n \times n}, \boldsymbol{B} \in \mathbb{R}^{n \times n}, \boldsymbol{C} \in \mathbb{R}^{d \times n}, \bar{\boldsymbol{y}} \in \mathbb{R}^{d}$,
        initialization $\boldsymbol{x}_0 \in \mathbb{R}^n$, sequence length $N_{\text{synth}} \in \mathbb{N}$
1 for $i = 1, \ldots, N_{\text{synth}}$ do
2     $\boldsymbol{v}_i \leftarrow \mathcal{N}(0, \boldsymbol{I}_n)$ ;            // Sample from Gaussian Noise
3     $\boldsymbol{x}_i \leftarrow \boldsymbol{A}\boldsymbol{x}_{i-1} + \boldsymbol{B}\boldsymbol{v}_i$;
4     $\boldsymbol{y}_i \leftarrow \boldsymbol{C}\boldsymbol{x}_i + \bar{\boldsymbol{y}}$;
5 end
Output : Synthesized sequence $\boldsymbol{y}_1, \ldots \boldsymbol{y}_{N_{\text{synth}}}$
```

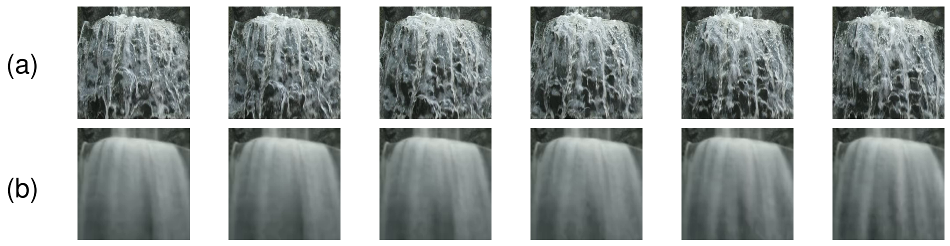


Figure 2.1.: Training (a) and synthesized (b) sequence of a visual process depicting a waterfall. Since synthesis has been performed using an LDS model, projection onto the principal subspace results in blurring of the video frames.

2.2. Video Synthesis with Linear Dynamic Systems

The learned parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ provide the capability to generate new, "synthetic" sequences of the visual process from scratch [38]. Synthesis of dynamic textures aims at recreating a video sequence without copying individual frames. Ideally, a synthesis algorithm is capable of producing video sequences of infinite length without introducing any visible discontinuities [130].

A synthesis algorithm employing an LDS model is described in Algorithm 3. To generate a sequence based on the LDS parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$, it suffices to generate a sequence of i.i.d. standard Gaussian noise samples in $\mathbb{R}^n$, and to feed them to the recursive model as the noise input term. Typical applications of video synthesis can be found in *Computer Generated Imagery* (CGI). However, generating unseen sequences of a visual process can come in handy in other areas as well. For instance, Algorithm 3 demonstrates that a video sequence can be represented by a model parameter tuple in combination with a point sequence in $\mathbb{R}^n$. Such a compact representation can aid in restoration and compression.

Figure 2.1 shows an example result of Algorithm 3. It is clearly visible that the synthesis procedure produces very blurred out frames, which is due to them being constrained to a low-dimensional subspace of $\mathbb{R}^d$. Nevertheless, the simplicity of Algorithm 3 is a strong argument in favor of it. Unlike many other synthesis procedures [127, 119], each frame can be generated on-line, recursively from the previous one, so that the computational burden increases only linearly with the sequence length. Later in this thesis, we will see how to keep this advantage while improving the visual quality.

2.3. LDS Parameters as Features

The extracted parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}$ contain distinguishing information about the visual process. While $\boldsymbol{C}$ and $\bar{\boldsymbol{y}}$ encode the appearance, the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ represent the temporal dynamics of the process. This can be exploited in problems such as classification or clustering of visual processes, where expressive features are demanded. To do so, a measure of similarity is required.

One important problem that arises in formulating a similarity measure on the system parameters is that the representation in Eq. (1.5) is not unique. In fact, a change of basis in the state space by means of an invertible matrix $\boldsymbol{P}$, i.e.,

$$\tilde{\boldsymbol{x}}_t = \boldsymbol{P}\boldsymbol{x}_t, \tag{2.24}$$

yields the representation

$$\begin{aligned} \tilde{\boldsymbol{x}}_{t+1} &= \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}}_t + \tilde{\boldsymbol{B}}\boldsymbol{v}_t, \\ \boldsymbol{y}_t &= \bar{\boldsymbol{y}} + \tilde{\boldsymbol{C}}\tilde{\boldsymbol{x}}_t, \end{aligned} \tag{2.25}$$

with

$$\tilde{\boldsymbol{A}} = \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{-1}, \qquad \tilde{\boldsymbol{B}} = \boldsymbol{P}\boldsymbol{B}, \qquad \tilde{\boldsymbol{C}} = \boldsymbol{C}\boldsymbol{P}^{-1}, \tag{2.26}$$

that describes a visual process with the same statistical properties. More precisely, a system with these parameters gives rise to a stochastic process with the same statistical moments. Let us denote the general linear Group of real invertible matrices in $\mathbb{R}^{n\times n}$ by $\mathrm{GL}(n)$. An LDS with the parameter tuple $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}})$ is statistically equivalent to all systems with parameter tuples in

$$\begin{aligned} &\mathcal{S}_{\mathrm{LDS}}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}) \\ =&\{(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{-1}, \boldsymbol{P}\boldsymbol{B}, \boldsymbol{C}\boldsymbol{P}^{-1}, \bar{\boldsymbol{y}}) \mid \boldsymbol{P} \in \mathrm{GL}(n)\}. \end{aligned} \tag{2.27}$$

The change of basis according to Eq. (2.26) is of considerable importance for the following chapters. For the sake of simplicity, let us introduce the notation

$$\boldsymbol{P} \bullet (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}) = (\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{-1}, \boldsymbol{P}\boldsymbol{B}, \boldsymbol{C}\boldsymbol{P}^{-1}, \bar{\boldsymbol{y}}), \tag{2.28}$$

to express this transformation in a more concise way.

A (dis)similarity measure should account for the ambiguity caused by transformations as described by Eq. (2.28). This is not necessarily the case for trivial distance measures. For instance, given the non-negative weighting parameters $\lambda_{\boldsymbol{A}}, \lambda_{\boldsymbol{B}}, \lambda_{\bar{\boldsymbol{y}}}$, a distance $d^2_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{y}}}}$ that measures a weighted sum of squared Frobenius norms between two LDS's $\Theta_1 = (\boldsymbol{A}_1, \boldsymbol{B}_1, \boldsymbol{C}_1, \bar{\boldsymbol{y}}_1)$ and $\Theta_2 = (\boldsymbol{A}_2, \boldsymbol{B}_2, \boldsymbol{C}_2, \bar{\boldsymbol{y}}_2)$, defined as

$$\begin{aligned} d^2_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{y}}}}(\Theta_1, \Theta_2) = & \|\boldsymbol{C}_1 - \boldsymbol{C}_2\|_F^2 + \lambda_{\boldsymbol{A}}\|\boldsymbol{A}_1 - \boldsymbol{A}_2\|_F^2 \\ & + \lambda_{\boldsymbol{B}}\|\boldsymbol{B}_1 - \boldsymbol{B}_2\|_F^2 + \lambda_{\bar{\boldsymbol{y}}}\|\bar{\boldsymbol{y}}_1 - \bar{\boldsymbol{y}}_2\|_F^2, \end{aligned} \tag{2.29}$$

is not a good a choice in general. To see this, consider the case $\Theta_2 = \boldsymbol{P} \bullet \Theta_1$ for some full-rank matrix $\boldsymbol{P} \neq \boldsymbol{I}_n$. Then, using the parameter values $\lambda_{\boldsymbol{A}} = 0, \lambda_{\boldsymbol{B}} = 0, \lambda_{\bar{\boldsymbol{y}}} = 0$ yields

$$d^2_{F,0,0,0}(\Theta_1, \Theta_2) = \|\boldsymbol{C}_1 - \boldsymbol{C}_2\boldsymbol{P}^{-1}\|_F^2. \tag{2.30}$$

This term is strictly positive, which should not happen, as Θ_1 and Θ_2 describe statistically equivalent systems. A good distance should at the very least treat equivalent systems equally. This is reflected in the following definition.

Definition 1. *Let d be a symmetric distance measure defined on LDS parameter tuples, i.e.,*

$$d_{\text{binv}}(\Theta_1, \Theta_2) = d_{\text{binv}}(\Theta_2, \Theta_1) \geq 0 \tag{2.31}$$

and

$$d_{\text{binv}}(\Theta_1, \Theta_1) = 0 \tag{2.32}$$

holds for any two systems Θ_1, Θ_2. We call d_{binv} basis-invariant*, if it is not altered by changes of basis in the state space, i.e. if the condition*

$$d_{\text{binv}}(\Theta_1, \Theta_2) = d_{\text{binv}}(\Theta_1, \boldsymbol{P} \bullet \Theta_2) \qquad \forall\, \boldsymbol{P} \in \mathrm{GL}(n), \tag{2.33}$$

is fulfilled.

Note that a basis-invariant distance d_{binv} vanishes if its two arguments are statistically equivalent, i.e.

$$d_{\text{binv}}(\Theta, \boldsymbol{P} \bullet \Theta) = 0 \qquad \forall\, \boldsymbol{P} \in \mathrm{GL}(n). \tag{2.34}$$

In the following, three basis-invariant distance measures that will be required later on are discussed. To facilitate this discussion, let us first formally define the set we are operating on.

When we use Algorithm 2 to extract a parameter tuple from a sequence of a visual process, we can assume the following two properties about it. First, all singular values of $\boldsymbol{A}$ are smaller than one, and second, $\boldsymbol{C}$ has orthonormal columns. Note that the latter assumption does not actually reduce the set of visual processes due to the state space invariance. Furthermore, the former assumption is sensible, as unstable systems are of little practical relevance to us. We thus define the set of feasible LDS's as

$$\mathcal{O}_d^n = \{(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \bar{\boldsymbol{y}}) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathrm{St}(n,d) \times \mathbb{R}^d \mid \|\boldsymbol{A}\|_2 < 1\}. \tag{2.35}$$

Since $\boldsymbol{C}$ is now restricted to the Stiefel manifold, a transformation of the form in Eq. (2.28) produces only a tuple in $\mathcal{O}_d^n$, if $\boldsymbol{P}$ is orthogonal. With $\mathrm{O}(n)$ denoting the *orthogonal group* in $\mathbb{R}^{n\times n}$, we define thus the relation

$$\mathcal{Q}_d^n = \{(\Theta_1, \Theta_2) \in \mathcal{O}_d^n \times \mathcal{O}_d^n \mid \exists \boldsymbol{Q} \in \mathrm{O}(n) \text{ s.t. } \Theta_1 = \boldsymbol{Q} \bullet \Theta_2\}. \tag{2.36}$$

For the quotient space induced by this relation, we write

$$\mathcal{QO}_d^n = \mathcal{O}_d^n / \mathcal{Q}_d^n = \{\{\boldsymbol{Q} \bullet \Theta \mid \boldsymbol{Q} \in \mathrm{O}(n)\} \mid \Theta \in \mathcal{O}_d^n\}. \tag{2.37}$$

Ideally, we should be able to interpret an LDS distance as a dissimilarity measure on the elements of $\mathcal{QO}_d^n$, since every pair of parameter tuples that are equivalent according to Eq. (2.36) should be treated equally. In what follows, we discuss three possible candidates that fulfill this requirement.

Martin Distance

The *Martin distance* [32] is derived from the *principal angles* between the *observation ranges* of two LDS's. To better understand what is meant by this conception, let us consider two systems

$$\Theta_1 = (\boldsymbol{A}_1, \boldsymbol{B}_1, \boldsymbol{C}_1, \bar{\boldsymbol{y}}_1),\ \Theta_2 = (\boldsymbol{A}_2, \boldsymbol{B}_2, \boldsymbol{C}_2, \bar{\boldsymbol{y}}_2) \in \mathcal{O}_d^n. \tag{2.38}$$

The Martin distance neglects the process noise and the observation offset, treating them as non-existent. This means that the parameters $\boldsymbol{B}_1, \boldsymbol{B}_2, \bar{\boldsymbol{y}}_1, \bar{\boldsymbol{y}}_2$ are treated as 0.

One way to motivate the Martin distance between Θ_1 and Θ_2 is to compare the possible trajectories in the observation space. To this end, let us again consider the VAR model in Eq. (1.5a), while substituting the parameters of Θ_1 and Θ_2, respectively. We assume that $\boldsymbol{B}_1 = \boldsymbol{B}_2 = 0$ holds, and denote the respective initial state by $\boldsymbol{x}_0^a$ for $a \in \{1,2\}$. Then, the VAR model produces the entirely deterministic sequence

$$\left(\boldsymbol{A}_a^k \boldsymbol{x}_0^a\right)_{k\in\mathbb{N}} = \left(\boldsymbol{x}_0^a,\ \boldsymbol{A}_a\boldsymbol{x}_0^a,\ \boldsymbol{A}_a^2\boldsymbol{x}_0^a,\ \ldots\right), \qquad a \in \{1,2\}. \tag{2.39}$$

Since $\bar{\boldsymbol{y}}_1$ and $\bar{\boldsymbol{y}}_2$ are also neglected, the corresponding observation can be obtained by multiplying the sequence members by the respective observation matrix. This yields what we refer to as the system *trajectory sequence*

$$\begin{aligned} T_{\Theta_a}(\boldsymbol{x}_0^a) &= (\boldsymbol{y}_k^a)_{k\in\mathbb{N}} \\ &= (\boldsymbol{y}_0^a,\ \boldsymbol{y}_1^a,\ \boldsymbol{y}_2^a,\ \ldots) \\ &= \left(\boldsymbol{C}_a\boldsymbol{x}_0^a,\ \boldsymbol{C}_a\boldsymbol{A}_a\boldsymbol{x}_0^a,\ \boldsymbol{C}_a\boldsymbol{A}_a^2\boldsymbol{x}_0^a,\ \ldots\right), \qquad a \in \{1,2\}. \end{aligned} \tag{2.40}$$

Note that the inequality

$$\|\boldsymbol{A}_a^k \boldsymbol{x}_0^a\| \leq \|\boldsymbol{A}_a\|_2^k \|\boldsymbol{x}_0^a\| \tag{2.41}$$

is always fulfilled, and that $\mathcal{O}_d^n$ is defined in such a way that $\|\boldsymbol{A}_a\|_2$ is always smaller than one. This means that the sum of squared Euclidean norms applied to the members of a trajectory sequence always exists, i.e.,

$$\begin{aligned} \|T_{\Theta_a}(\boldsymbol{x}_0^a)\|^2 =& \sum_{k=1}^{\infty} \|\boldsymbol{C}_a \boldsymbol{A}_a^k \boldsymbol{x}_0^a\|^2 \\ =& \sum_{k=1}^{\infty} \|\boldsymbol{A}_a^k \boldsymbol{x}_0^a\|^2 \\ \leq& \|\boldsymbol{x}_0^a\|^2 \sum_{k=1}^{\infty} \|\boldsymbol{A}_a\|_2^{2k} < \infty \end{aligned} \tag{2.42}$$

is satisfied. In fact, if we rewrite $T_{\Theta_a}(\boldsymbol{x}_0^a)$ as a sequence of the scalar elements of its members, the result will belong to ℓ_2, the Hilbert space of quadratically summable sequences. Moreover, the set

$$\mathcal{T}(\Theta_a) = \{T_{\Theta_a}(\boldsymbol{x}_0^a) \mid \boldsymbol{x}_0^a \in \mathbb{R}^n\} \tag{2.43}$$

can be interpreted as an n-dimensional subspace of ℓ_2. For $a, b \in \{1, 2\}$, let us now define the matrix

$$\boldsymbol{\Phi}_{a,b} = \sum_{k=0}^{\infty} \boldsymbol{A}_a^{k\top} \boldsymbol{C}_a^\top \boldsymbol{C}_b \boldsymbol{A}_b^k. \tag{2.44}$$

Analogously to Eq. (2.42), this sum always exists so that the canonical inner product between two trajectory sequences $T_{\Theta_a}(\boldsymbol{x}_0^a), T_{\Theta_b}(\boldsymbol{x}_0^b)$ can be computed as

$$\begin{aligned} \left\langle T_{\Theta_a}(\boldsymbol{x}_0^a), T_{\Theta_b}(\boldsymbol{x}_0^b) \right\rangle_{\ell_2} =& \sum_{k=0}^{\infty} \boldsymbol{y}_k^{a\top} \boldsymbol{y}_k^b \\ =& \sum_{k=0}^{\infty} \boldsymbol{x}_0^{a\top} \boldsymbol{A}_a^{k\top} \boldsymbol{C}_a^\top \boldsymbol{C}_b \boldsymbol{A}_b^k \boldsymbol{x}_0^b \\ =& \boldsymbol{x}_0^{a\top} \boldsymbol{\Phi}_{a,b} \boldsymbol{x}_0^b. \end{aligned} \tag{2.45}$$

For two systems $\Theta_1, \Theta_2 \in \mathcal{O}_d^n$, the Martin distance is essentially based on a measure of similarity between the spaces $\mathcal{T}(\Theta_1)$ and $\mathcal{T}(\Theta_2)$. To this end, the normalized squared scalar product

$$\frac{\langle T_{\Theta_1}(\boldsymbol{x}_0^1), T_{\Theta_2}(\boldsymbol{x}_0^2)\rangle_{\ell_2}^2}{\langle T_{\Theta_1}(\boldsymbol{x}_0^1), T_{\Theta_1}(\boldsymbol{x}_0^1)\rangle_{\ell_2} \langle T_{\Theta_2}(\boldsymbol{x}_0^2), T_{\Theta_2}(\boldsymbol{x}_0^2)\rangle_{\ell_2}} = \frac{\left(\boldsymbol{x}_0^{1\top} \boldsymbol{\Phi}_{1,2} \boldsymbol{x}_0^2\right)^2}{\left(\boldsymbol{x}_0^{1\top} \boldsymbol{\Phi}_{1,1} \boldsymbol{x}_0^1\right) \left(\boldsymbol{x}_0^{2\top} \boldsymbol{\Phi}_{2,2} \boldsymbol{x}_0^2\right)} \tag{2.46}$$

is employed as a similarity measure between the two trajectories $T_{\Theta_1}(\boldsymbol{x}_0^1)$ and $T_{\Theta_2}(\boldsymbol{x}_0^2)$.

Consider two orthogonal bases $\{T_{\Theta_1}(\boldsymbol{x}_0^{1,i})\}_{i,\dots,n}$ and $\{T_{\Theta_2}(\boldsymbol{x}_0^{2,i})\}_{i,\dots,n}$ such that the term in Eq. (2.46) is maximized for each pair $\boldsymbol{x}_0^{1,i}, \boldsymbol{x}_0^{2,i}$. The Martin distance is computed by summing up the logarithm of the resulting terms. This problem is solved iteratively. To do so, for each $i \in \{1, \dots, n\}$, the optimization problem

$$\hat{\boldsymbol{x}}_0^{1,i}, \hat{\boldsymbol{x}}_0^{2,i} = \underset{\boldsymbol{x}_0^{1,i}, \boldsymbol{x}_0^{2,i}}{\arg\max} \frac{\left(\boldsymbol{x}_0^{1,i\top} \boldsymbol{\Phi}_{1,2} \boldsymbol{x}_0^{2,i}\right)^2}{\left(\boldsymbol{x}_0^{1,i\top} \boldsymbol{\Phi}_{1,1} \boldsymbol{x}_0^{1,i}\right)\left(\boldsymbol{x}_0^{2,i\top} \boldsymbol{\Phi}_{2,2} \boldsymbol{x}_0^{2,i}\right)} \tag{2.47}$$

is solved, such that for all $j < i$, the condition

$$\boldsymbol{x}_0^{1,i\top} \boldsymbol{\Phi}_{1,1} \hat{\boldsymbol{x}}_0^{1,j} = 0, \ \boldsymbol{x}_0^{2,i\top} \boldsymbol{\Phi}_{2,2} \hat{\boldsymbol{x}}_0^{2,j} = 0, \tag{2.48}$$

is satisfied. In each iteration i, this yields two basis vectors $T_{\Theta_1}(\boldsymbol{x}_0^{1,i})$ and $T_{\Theta_2}(\boldsymbol{x}_0^{2,i})$, such that they are orthogonal to all the previously computed basis vectors of the respective subspaces in ℓ_2, and the normalized squared inner product between them is maximized.

The Martin distance is the sum

$$d_{\text{Martin}}(\Theta_1, \Theta_2) = -\sum_{i=1}^{n} \log \frac{\left(\hat{\boldsymbol{x}}_0^{1,i\top} \boldsymbol{\Phi}_{1,2} \hat{\boldsymbol{x}}_0^{2,i}\right)^2}{\left(\hat{\boldsymbol{x}}_0^{1,i\top} \boldsymbol{\Phi}_{1,1} \hat{\boldsymbol{x}}_0^{1,i}\right)\left(\hat{\boldsymbol{x}}_0^{2,i\top} \boldsymbol{\Phi}_{2,2} \hat{\boldsymbol{x}}_0^{2,i}\right)}. \tag{2.49}$$

In essence, determining Eq. (2.49) for two systems requires the computation of $\boldsymbol{\Phi}_{1,1}, \boldsymbol{\Phi}_{1,2}, \boldsymbol{\Phi}_{2,2}$ and solving Eq. (2.47).

The computation of $\boldsymbol{\Phi}_{a,b}, \ a, b \in \{1, 2\}$ does not need to be carried out numerically. Rather, multiplying $\boldsymbol{\Phi}_{a,b}$ as defined in Eq. (2.44) by $\boldsymbol{A}_a^\top$ from the left and $\boldsymbol{A}_b$ from the right yields the *discrete Sylvester equation* [13], i.e.,

$$\boldsymbol{A}_a^\top \boldsymbol{\Phi}_{a,b} \boldsymbol{A}_b + \boldsymbol{C}_a^\top \boldsymbol{C}_b = \boldsymbol{\Phi}_{a,b}. \tag{2.50}$$

Eq. (2.50) can be analytically solved by vectorizing $\boldsymbol{\Phi}_{a,b}$, i.e., writing

$$(\boldsymbol{A}_b \otimes \boldsymbol{A}_a) \operatorname{vec}(\boldsymbol{\Phi}_{a,b}) + \operatorname{vec}(\boldsymbol{C}_a^\top \boldsymbol{C}_b) = \operatorname{vec}(\boldsymbol{\Phi}_{a,b}) \tag{2.51}$$

and reformulating as

$$\operatorname{vec}(\boldsymbol{\Phi}_{a,b}) = (\boldsymbol{I}_{n^2} - \boldsymbol{A}_b \otimes \boldsymbol{A}_a)^{-1} \operatorname{vec}(\boldsymbol{C}_a^\top \boldsymbol{C}_b). \tag{2.52}$$

As for Eq. (2.47), it can be solved by fixing $\boldsymbol{z}^{a,i} = \boldsymbol{\Phi}_{a,a}^{1/2} {\boldsymbol{x}_0}^{a,i}, a \in \{1, 2\}$. Then, it can be rewritten as

$$\hat{\boldsymbol{z}}^{1,i}, \hat{\boldsymbol{z}}^{2,i} = \underset{\boldsymbol{z}^{1,i}, \boldsymbol{z}^{2,i}}{\arg\max} \frac{\left(\boldsymbol{z}^{1,i\top} \boldsymbol{\Phi}_{1,1}^{-1/2} \boldsymbol{\Phi}_{1,2} \boldsymbol{\Phi}_{2,2}^{-1/2} \boldsymbol{z}^{2,i}\right)^2}{\|\boldsymbol{z}^{1,i}\|^2 \|\boldsymbol{z}^{2,i}\|^2}, \tag{2.53}$$

Algorithm 4 : Computation of the Martin Distance

```
  Input : Systems
          $\Theta_1 = (\boldsymbol{A}_1, \boldsymbol{B}_1, \boldsymbol{C}_1, \bar{\boldsymbol{y}}_1) \in \mathcal{O}_d^n,\ \Theta_2 = (\boldsymbol{A}_2, \boldsymbol{B}_2, \boldsymbol{C}_2, \bar{\boldsymbol{y}}_2) \in \mathcal{O}_d^n$
1 Compute matrices $\boldsymbol{\Phi}_{1,1}, \boldsymbol{\Phi}_{1,2}, \boldsymbol{\Phi}_{2,2}$;                // Eq. (2.52)
2 Compute singular values $\sigma_i$ of $\boldsymbol{\Phi}_{1,1}^{-1/2}\boldsymbol{\Phi}_{1,2}\boldsymbol{\Phi}_{2,2}^{-1/2}$;
3 $d_{\text{Martin}}(\Theta_1, \Theta_2) \leftarrow -2\sum_{i=1}^{n} \log \sigma_i$;
  Output : Martin distance $d_{\text{Martin}}(\Theta_1, \Theta_2)$
```

while the condition in Eq. (2.48) can be expressed as

$$\boldsymbol{z}^{1,i\top}\hat{\boldsymbol{z}}^{1,j} = 0,\ \boldsymbol{z}^{2,i\top}\hat{\boldsymbol{z}}^{2,j} = 0, \qquad \forall i > j. \tag{2.54}$$

This problem is solved by the left and right singular vectors of $\boldsymbol{\Phi}_{1,1}^{-1/2}\boldsymbol{\Phi}_{1,2}\boldsymbol{\Phi}_{2,2}^{-1/2}$, respectively. Eq. (2.49) can thus be rewritten as

$$d_{\text{Martin}}(\Theta_1, \Theta_2) = -2\sum_{i=1}^{n} \log \sigma_i, \tag{2.55}$$

where σ_i denotes the ith singular value of $\boldsymbol{\Phi}_{1,1}^{-1/2}\boldsymbol{\Phi}_{1,2}\boldsymbol{\Phi}_{2,2}^{-1/2}$.

While being a basis-invariant dissimilarity measure, the Martin distance has a disadvantage. As it does not take into account the observation offset $\bar{\boldsymbol{y}}$, it tends to miss essential characteristics of the visual processes.

Maximum Singular Value

Consider again Eq. (2.50). In order to compute the *maximum singular value* (Max SV) distance [28], an additional tuning parameter $0 < \lambda < 1$ is introduced as

$$\lambda \boldsymbol{A}_a^\top \tilde{\boldsymbol{\Phi}}_{a,b}\boldsymbol{A}_b + \boldsymbol{C}_a^\top \boldsymbol{C}_b = \tilde{\boldsymbol{\Phi}}_{a,b}. \tag{2.56}$$

Given the solutions for $\tilde{\boldsymbol{\Phi}}_{1,1}, \tilde{\boldsymbol{\Phi}}_{1,2}, \tilde{\boldsymbol{\Phi}}_{2,2}$, the Max SV distance is defined as

$$d_{\text{MaxSV},\lambda}(\Theta_1, \Theta_2) = 2\left(1 - \frac{\|\tilde{\boldsymbol{\Phi}}_{1,2}\|_2}{\sqrt{\|\tilde{\boldsymbol{\Phi}}_{1,1}\|_2\|\tilde{\boldsymbol{\Phi}}_{2,2}\|_2}}\right). \tag{2.57}$$

It has been previously employed in human action recognition [28]. Note that it has the same drawback with regards to the observation offset as the Martin distance.

Alignment Distance

The *Alignment distance* was introduced in [2] and is based on the weighted Frobenius distance of the parameters $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$. We define it here in a slightly adapted way, by also including the parameter $\bar{\boldsymbol{y}}$. For two systems Θ_1 and Θ_2, we write its squared form as

$$\begin{aligned} d^2_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{y}}}}(\Theta_1,\Theta_2) &= \min_{\boldsymbol{Q}\in\mathrm{O}(n)} d^2_{\mathrm{F},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{y}}}}(\Theta_1,\boldsymbol{Q}\bullet\Theta_2) \\ &= \min_{\boldsymbol{Q}\in\mathrm{O}(n)} \|\boldsymbol{C}_1-\boldsymbol{C}_2\boldsymbol{Q}^\top\|_F^2 \\ &\qquad + \lambda_{\boldsymbol{A}}\|\boldsymbol{A}_1-\boldsymbol{Q}\boldsymbol{A}_2\boldsymbol{Q}^\top\|_F^2 \\ &\qquad + \lambda_{\boldsymbol{B}}\|\boldsymbol{B}_1-\boldsymbol{Q}\boldsymbol{B}_2\|_F^2+\lambda_{\bar{\boldsymbol{y}}}\|\bar{\boldsymbol{y}}_1-\bar{\boldsymbol{y}}_2\|^2. \end{aligned} \tag{2.58}$$

In order to compute the Alignment distance, it is necessary to find the orthogonal matrix $\boldsymbol{Q}$ that solves the optimization problem in Eq. (2.58) for $\boldsymbol{Q}$. This could be approached by a gradient descent method [1, 2], for instance.

2.4. Semi-linear Ambiguities

Previously, we have discussed ambiguities of models described by Eq. (1.5) with regards to the latent state space basis. It is important to be aware that these types of ambiguities carry over to the semi-linear case, when we replace Eq. (1.5b) by Eq. (1.6b).

Given a function $\Gamma\colon \mathbb{R}^n \to \mathbb{R}^d$ and a full-rank matrix $\boldsymbol{P}\in\mathbb{R}^{n\times n}$, we write $\Gamma\circ\boldsymbol{P}^{-1}$ to denote a composition of a left-side multiplication by $\boldsymbol{P}^{-1}$ with Γ. Then, given a semi-linear system described by Eq. (1.6) with the parameters $\boldsymbol{A},\boldsymbol{B},\Gamma$, all parameter tuples in

$$\mathcal{S}_{\mathrm{SLDS}}(\boldsymbol{A},\boldsymbol{B},\Gamma)=\{(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{-1},\boldsymbol{P}\boldsymbol{B},\Gamma\circ\boldsymbol{P}^{-1},\bar{\boldsymbol{y}})\mid \boldsymbol{P}\in\mathrm{GL}(n)\} \tag{2.59}$$

describe systems following the same statistical laws. This property will prove important in defining dissimilarity metrics and designing generative models for semi-linear systems.

2.5. Kernel Methods

Often, the assumption of a low-dimensional, affine subspace necessary for PCA to work cannot be upheld. In that case, a remedy could be to find a non-linear transformation that maps all the data points to a representation which does fulfill this assumption. *Kernel PCA* [108] is motivated by this very idea, while having one important property: It does not require an explicit form of the transformation. Instead, it suffices to know an explicit way to express the *inner product* of a pair of points in the transformed space. *Kernels* allow us to implicitly map pairs of data points to a possibly unknown Hilbert space, and to compute an inner product in this space.

Much has been written about the theory of kernels and reproducing kernel Hilbert spaces [109]. In what follows, the bulk of available theory is omitted and only the basic insights necessary for kernel PCA and *kernelized linear dynamic systems* are discussed.

Hilbert Spaces

While theoretical foundations of Hilbert spaces are not necessarily required for introducing kernel PCA, they can help to establish certain concepts that will come in handy later on. This section reviews some basic terminology and properties of Hilbert spaces. First, the notion of *separable* Hilbert spaces is discussed. Afterwards, matrices in finite-dimensional Euclidean spaces are generalized to possibly infinite dimensional Hilbert spaces. Finally, *Riesz' representation theorem* [33], a key result about Hilbert spaces is briefly described.

Recall that an *inner product* $\langle\cdot,\cdot\rangle_{\mathbb{V}}$ on a real-valued vector space $\mathbb{V}$ is a mapping

$$\langle\cdot,\cdot\rangle_{\mathbb{V}} : \mathbb{V} \times \mathbb{V} \to \mathbb{R}, \tag{2.60}$$

that is bi-linear, symmetric and positive definite. A *norm* $\|\cdot\|_{\mathbb{V}}$ is induced by an inner product $\langle\cdot,\cdot\rangle_{\mathbb{V}}$ as

$$\begin{aligned} \|\cdot\|_{\mathbb{V}} \colon \mathbb{V} &\to \mathbb{R}, \\ \boldsymbol{x} &\mapsto \sqrt{\langle \boldsymbol{x}, \boldsymbol{x}\rangle_{\mathbb{V}}}. \end{aligned} \tag{2.61}$$

A *Hilbert space* $\mathbb{H}$ (over $\mathbb{R}$) is a (real-valued) *vector space*, equipped with an *inner product* $\langle\cdot,\cdot\rangle_{\mathbb{H}}$, such that $\mathbb{H}$ is *complete* with regards to the norm $\|\cdot\|_{\mathbb{H}}$ induced by $\langle\cdot,\cdot\rangle_{\mathbb{H}}$. That means that every sequence in $\mathbb{H}$ that is Cauchy-convergent with regards to $\|\cdot\|_{\mathbb{H}}$ converges to a point in $\mathbb{H}$.

Let $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N\} \in \mathbb{H}$ be an orthonormal system in $\mathbb{H}$, i.e.,

$$\langle \boldsymbol{b}_i, \boldsymbol{b}_j\rangle_{\mathbb{H}} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall i, j \in \{1, \ldots, N\}. \tag{2.62}$$

Then, in analogy to $\mathbb{R}^n$, the operator

$$\begin{aligned} \Pi_{\text{span}\{\boldsymbol{b}_1,\ldots,\boldsymbol{b}_N\}} : \mathbb{H} &\to \mathbb{H}, \\ \boldsymbol{x} &\mapsto \sum_{i=1}^{N} \langle \boldsymbol{b}_i, \boldsymbol{x}\rangle_{\mathbb{H}} \boldsymbol{b}_i, \end{aligned} \tag{2.63}$$

describes an *orthogonal projection* onto $\text{span}\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N\}$. It is the operation that maps $\boldsymbol{x}$ to a point in the subspace of $\mathbb{H}$ spanned by the orthonormal system $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N\}$, such that the norm $\|\cdot\|_{\mathbb{H}}$ of the difference between the point and

$\boldsymbol{x}$ is minimized. We call a Hilbert space *separable*, if it possesses a countable *orthonormal basis*. This is straight-forward, if the Hilbert space is finite-dimensional. For an infinite-dimensional Hilbert space, this entails that there exists a sequence $(\boldsymbol{b}_i)_{i\in\mathbb{N}}$ in $\mathbb{H}$, such that any finite set of members of the sequence is an orthonormal system, and for any $\boldsymbol{x} \in \mathbb{H}$ and any real positive bound $\epsilon > 0$, an integer $N \in \mathbb{N}$ can be found, such that the inequality

$$\left\|\boldsymbol{x} - \Pi_{\operatorname{span}\{\boldsymbol{b}_1,\dots,\boldsymbol{b}_{N'}\}}\boldsymbol{x}\right\|_{\mathbb{H}} < \epsilon \tag{2.64}$$

holds for all $N' \geq N$. All Hilbert spaces in this thesis are assumed to be separable.

Let us fix the notation

$$\mathbb{H}^m = \underbrace{\mathbb{H} \times \cdots \times \mathbb{H}}_{m \text{ times}} \tag{2.65}$$

for Cartesian products of a space $\mathbb{H}$. To simplify notation, some parts of this thesis require a notion of Hilbert space *matrices* as a generalization of matrices in $\mathbb{R}^{m\times n}$. For this purpose, it is sensible to define a matrix $\boldsymbol{F} = \left[\boldsymbol{f}_1, \dots, \boldsymbol{f}_m\right] \in \mathbb{H}^m$ as an m-dimensional tuple of vectors in a separable Hilbert space $\mathbb{H}$. We use the notation $(\cdot)_{i,j}$ for the element in ith row and jth column of a matrix in $\mathbb{R}^{m\times n}$ and define multiplication with a matrix $\boldsymbol{D} \in \mathbb{R}^{m\times n}$ from the right as

$$\boldsymbol{F}\boldsymbol{D} = \left[\sum_{i=1}^m (\boldsymbol{D})_{1,i}\boldsymbol{f}_i \quad \cdots \quad \sum_{i=1}^m (\boldsymbol{D})_{n,i}\boldsymbol{f}_i\right] \in \mathbb{H}^n. \tag{2.66}$$

For two matrices $\boldsymbol{F} \in \mathbb{H}^m, \boldsymbol{G} \in \mathbb{H}^n$ we can also introduce the notion of a matrix product

$$\boldsymbol{F}^\top\boldsymbol{G} = \begin{bmatrix} \langle \boldsymbol{f}_1, \boldsymbol{g}_1\rangle_{\mathbb{H}} & \cdots & \langle \boldsymbol{f}_1, \boldsymbol{g}_n\rangle_{\mathbb{H}} \\ \vdots & \ddots & \vdots \\ \langle \boldsymbol{f}_m, \boldsymbol{g}_1\rangle_{\mathbb{H}} & \cdots & \langle \boldsymbol{f}_m, \boldsymbol{g}_n\rangle_{\mathbb{H}} \end{bmatrix} \in \mathbb{R}^{m\times n}. \tag{2.67}$$

In particular, the scalar product between to vectors $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{H}$ in a Hilbert space will be also written as

$$\boldsymbol{h}_1^\top\boldsymbol{h}_2 = \langle \boldsymbol{h}_1, \boldsymbol{h}_2\rangle_{\mathbb{H}}. \tag{2.68}$$

Consequently, the Frobenius norm on matrices in $\mathbb{H}^n$, is defined as

$$\begin{aligned} \|\cdot\|_F \colon \mathbb{H}^n &\to \mathbb{R} \\ \boldsymbol{F} &\mapsto \sqrt{\operatorname{tr}(\boldsymbol{F}^\top\boldsymbol{F})}. \end{aligned} \tag{2.69}$$

One important property of Hilbert spaces is *Riesz' representation theorem* [33] which states that for any linear, bounded functional $\gamma\colon \mathbb{H} \to \mathbb{R}$ from a Hilbert space $\mathbb{H}$ to the space of real numbers, there is a vector $\boldsymbol{h} \in \mathbb{H}$ such that it can be written as

$$\gamma(\boldsymbol{x}) = \boldsymbol{h}^\top\boldsymbol{x}, \qquad \forall \boldsymbol{x} \in \mathbb{H}. \tag{2.70}$$

This property is not of critical importance to this thesis, but it helps to recall it, in order to understand the functionality of kernel based representation learning.

The Kernel Trick

Let us once again consider a matrix $\boldsymbol{Y} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_N]$ of N data points in $\mathbb{R}^d$. This time, we do not assume that the data points are gathered around an affine subspace of $\mathbb{R}^d$. Instead, we act on the assumption that there is a non-linear mapping $\phi\colon \mathbb{R}^d \to \mathbb{H}$ that maps the data points to a separable Hilbert space $\mathbb{H}$, in which the transformed data points $\phi(\boldsymbol{y}_1), \dots, \phi(\boldsymbol{y}_N)$ do fulfill this property. Then, we could try to find an affine subspace of $\mathbb{H}$ in order to obtain a low-dimensional parameterization of the data points. Unfortunately, it is very difficult to find an appropriate function ϕ that fulfills the desired properties. However, by employing the *kernel trick*, it is still possible to find a low-dimensional parameterization of $\boldsymbol{Y}$ in some Hilbert space, even when we do not have an explicit expression for ϕ. All it takes is a *kernel*. In the scope of this thesis, we define a kernel as a continuous function

$$\kappa\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \tag{2.71}$$

such that for *any* N data points $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ in $\mathbb{R}^d$, the *Gram matrix*

$$\boldsymbol{K} = \begin{bmatrix} \kappa(\boldsymbol{y}_1, \boldsymbol{y}_1) & \cdots & \kappa(\boldsymbol{y}_1, \boldsymbol{y}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{y}_N, \boldsymbol{y}_1) & \cdots & \kappa(\boldsymbol{y}_N, \boldsymbol{y}_N) \end{bmatrix} \tag{2.72}$$

is symmetric and positive semi-definite. It can be shown that for every kernel $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, there is a mapping ϕ from $\mathbb{R}^d$ to some separable Hilbert space $\mathbb{H}$, such that it can be expressed as

$$\kappa(\boldsymbol{y}_1, \boldsymbol{y}_2) = \langle \phi(\boldsymbol{y}_1), \phi(\boldsymbol{y}_2) \rangle_{\mathbb{H}} \tag{2.73}$$

for all $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^d$. In that case, we call $\mathbb{H}$ a *kernel feature space* and ϕ a *feature space mapping* induced by κ.

It is difficult to say when a kernel corresponds to a feature space mapping that exhibits the desired property of linearizing the data. However, heuristics based on properties such as locality suggest a variety of kernels that are in use for all sorts of applications [18].

We will come back later to the question, what kernels are suitable for problems related to visual processes. For now, let us disregard the difficulty of finding the proper kernel, and assume that we have found a κ that meets our demands. The existence of such a kernel entails a vast improvement for a variety of machine learning techniques that we might want to apply to our data. The reason can be found in Riesz' representation theorem. Since any linear function can be expressed by means of an inner product, a linear machine learning algorithm can be enhanced by reformulating it in terms of inner products and then replacing all inner products by κ. Such a substitution enables the algorithm to work on a more appropriate representation of the data, rather than on the data itself.

Kernel PCA

With the kernel κ that we believe to be an appropriate choice for our data $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N \in \mathbb{R}^d$, we write

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi(\boldsymbol{y}_1) & \cdots & \phi(\boldsymbol{y}_N) \end{bmatrix} \in \mathbb{H}^N, \tag{2.74}$$

where $\phi : \mathbb{R}^d \to \mathbb{H}$ denotes a feature space mapping implicitly described by κ. Remember that an explicit form of $\boldsymbol{\Phi}$ usually does not exist. So what we want is a way to find a PCA-like low-dimensional parameterization of $\boldsymbol{\Phi}$ that does not require an explicit expression for ϕ. Analogously to standard PCA and Eq. (2.2) and Eq. (2.3), we are looking for an affine mapping

$$\boldsymbol{\varphi} \mapsto \boldsymbol{F}^\top(\boldsymbol{\varphi} + \boldsymbol{b}), \tag{2.75}$$

with $\boldsymbol{F} \in \mathbb{H}^n, \boldsymbol{b}, \boldsymbol{\varphi} \in \mathbb{H}$, which in combination with some complementary affine mapping

$$\boldsymbol{x} \mapsto \tilde{\boldsymbol{F}}\boldsymbol{x} + \boldsymbol{c}, \tag{2.76}$$

with $\tilde{\boldsymbol{F}} \in \mathbb{H}^n, \boldsymbol{c} \in \mathbb{H}$, minimizes loss

$$L_{\mathrm{KPCA}}(\boldsymbol{F}, \tilde{\boldsymbol{F}}, \boldsymbol{b}, \boldsymbol{c}) = \sum_{i=1}^{N} \left\| \tilde{\boldsymbol{F}}(\boldsymbol{F}^\top(\phi(\boldsymbol{y}_i) + \boldsymbol{b})) + \boldsymbol{c} - \phi(\boldsymbol{y}_i) \right\|_{\mathbb{H}}^2. \tag{2.77}$$

Eq. (2.77) describes the summed squared error between the data representations $\phi(\boldsymbol{y}_i)$ and their kernel PCA reconstructions that result from applying Eq. (2.75) and Eq. (2.76) to them. Like in a finite-dimensional Euclidean vector space, we can determine that

$$\tilde{\boldsymbol{F}}\boldsymbol{F}^\top\boldsymbol{b} + \boldsymbol{c} = -\tilde{\boldsymbol{F}}\boldsymbol{F}^\top \frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{y}_i) + \frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{y}_i) \tag{2.78}$$

needs to be satisfied for minimizing Eq. (2.77), by using the arithmetic mean. This directly yields

$$\boldsymbol{b} = -\boldsymbol{c} = -\frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{y}_i) = -\frac{1}{N}\boldsymbol{\Phi}\boldsymbol{1}_N \tag{2.79}$$

as a solution. Substituting this result into Eq. (2.77) yields

$$\begin{aligned} &L_{\mathrm{KPCA}}\left(\boldsymbol{F}, \tilde{\boldsymbol{F}}, -\frac{1}{N}\boldsymbol{\Phi}\boldsymbol{1}_N, \frac{1}{N}\boldsymbol{\Phi}\boldsymbol{1}_N\right) \\ =&\sum_{i=1}^{N}\left\| \tilde{\boldsymbol{F}}(\boldsymbol{F}^\top(\phi(\boldsymbol{y}_i) - \frac{1}{N}\boldsymbol{\Phi}\boldsymbol{1}_N)) + \frac{1}{N}\boldsymbol{\Phi}\boldsymbol{1}_N - \phi(\boldsymbol{y}_i)\right\|_{\mathbb{H}}^2. \end{aligned} \tag{2.80}$$

Subsequently, let us define

$$\boldsymbol{\Phi}^{\mathrm{c}} = \boldsymbol{\Phi} - \frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N\mathbf{1}_N^\top. \tag{2.81}$$

The n-dimensional subspace of $\mathbb{H}$ that we are looking for is contained in the column space of $\boldsymbol{\Phi}^{\mathrm{c}}$, which means that we can write $\boldsymbol{F} = \boldsymbol{\Phi}^{\mathrm{c}}\boldsymbol{R}$ and $\tilde{\boldsymbol{F}} = \boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}$ with $\boldsymbol{R}, \tilde{\boldsymbol{R}} \in \mathbb{R}^{N\times n}$. Substituting these definitions into Eq. (2.80) yields

$$\begin{aligned}
&L_{\mathrm{KPCA}}\left(\boldsymbol{\Phi}^{\mathrm{c}}\boldsymbol{R}, \boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}, -\frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N, \frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N\right)\\
=&\sum_{i=1}^{N}\|\boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}(\boldsymbol{R}^\top\boldsymbol{\Phi}^{\mathrm{c}\top}(\phi(\boldsymbol{y}_i) - \frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N)) + \frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N - \phi(\boldsymbol{y}_i)\|_{\mathbb{H}}^2\\
=&\|\boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{\Phi}^{\mathrm{c}\top}\boldsymbol{\Phi}^{\mathrm{c}} - \boldsymbol{\Phi}^{\mathrm{c}}\|_F^2.
\end{aligned} \tag{2.82}$$

Defining the centered Gram matrix

$$\boldsymbol{K}^{\mathrm{c}} = \boldsymbol{\Phi}^{\mathrm{c}\top}\boldsymbol{\Phi}^{\mathrm{c}} \tag{2.83}$$

allows us to rewrite Eq. (2.82) as

$$\begin{aligned}
&\|\boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{K}^{\mathrm{c}} - \boldsymbol{\Phi}^{\mathrm{c}}\|_F^2\\
=&\operatorname{tr}(\boldsymbol{K}^{\mathrm{c}}\boldsymbol{R}\tilde{\boldsymbol{R}}^\top\boldsymbol{\Phi}^{\mathrm{c}\top}\boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{K}^{\mathrm{c}} - 2\boldsymbol{\Phi}^{\mathrm{c}\top}\boldsymbol{\Phi}^{\mathrm{c}}\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{K}^{\mathrm{c}} + \boldsymbol{\Phi}^{\mathrm{c}\top}\boldsymbol{\Phi}^{\mathrm{c}})\\
=&\operatorname{tr}(\boldsymbol{K}^{\mathrm{c}}\boldsymbol{R}\tilde{\boldsymbol{R}}^\top\boldsymbol{K}^{\mathrm{c}}\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{K}^{\mathrm{c}} - 2\boldsymbol{K}^{\mathrm{c}}\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{K}^{\mathrm{c}} + \boldsymbol{K}^{\mathrm{c}}).
\end{aligned} \tag{2.84}$$

Due to symmetry of the centered Gram matrix $\boldsymbol{K}^{\mathrm{c}}$, it has an SVD of the form $\boldsymbol{K}^{\mathrm{c}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$. Furthermore, let us define

$$\boldsymbol{D} = \boldsymbol{U}^\top\tilde{\boldsymbol{R}}\boldsymbol{R}^\top\boldsymbol{U}. \tag{2.85}$$

We substitute this definition into Eq. (2.84) and obtain finally the optimization problem

$$\hat{\boldsymbol{D}} = \underset{\substack{\boldsymbol{D}\in\mathbb{R}^{N\times N},\\ \operatorname{rank}(\boldsymbol{D})\le n}}{\arg\min}\ \operatorname{tr}(\boldsymbol{\Lambda}\boldsymbol{D}^\top\boldsymbol{\Lambda}\boldsymbol{D}\boldsymbol{\Lambda} - 2\boldsymbol{\Lambda}\boldsymbol{D}\boldsymbol{\Lambda}). \tag{2.86}$$

The solution $\hat{\boldsymbol{D}}$ must be diagonal since off-diagonal elements contribute positively to the first term, but do not contribute at all to the second term. With this in mind, let us write the singular values of $\boldsymbol{K}^{\mathrm{c}}$ as $\lambda_1, \ldots, \lambda_n$, which yields

$$\hat{\boldsymbol{D}} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n, 0, \ldots, 0). \tag{2.87}$$

We can thus choose

$$\boldsymbol{R} = \tilde{\boldsymbol{R}} = \boldsymbol{U}_{(n)}\operatorname{diag}(\lambda_1, \ldots, \lambda_n)^{-1/2}. \tag{2.88}$$

Algorithm 5 : Kernel PCA

```
    Input : Data matrix $\boldsymbol{Y} \in \mathbb{R}^{d \times N}$, target dimension $n \in \mathbb{N}$
1   $\boldsymbol{K} \leftarrow \kappa(\boldsymbol{Y}, \boldsymbol{Y})$;                       // Computing Gram matrix
2   $\boldsymbol{K}^{\mathrm{c}} \leftarrow (\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top)\boldsymbol{K}(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top)$;        // Centering data
3   $\boldsymbol{U}, \boldsymbol{\Lambda}, \boldsymbol{U} \leftarrow \mathrm{SVD}(\boldsymbol{K}^{\mathrm{c}})$;
4   $\boldsymbol{R} \leftarrow \boldsymbol{U}_{(n)} \boldsymbol{\Lambda}_{(n)}^{(n)^{-1/2}}$;
    Output : Matrix $\boldsymbol{R}$
```

This leads us finally to a reformulation of Eq. (2.75) for computing the kernel PCA parameterization of an arbitrary feature space representation $\phi(\boldsymbol{y})$ of a data point $\boldsymbol{y} \in \mathbb{R}^d$. We can write it as

$$\begin{aligned} \boldsymbol{F}^\top(\phi(\boldsymbol{y}) + \boldsymbol{b}) &= \boldsymbol{R}^\top \boldsymbol{\Phi}^{\mathrm{c}\top} \left(\phi(\boldsymbol{y}) - \frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N \right) \\ &= \boldsymbol{R}^\top \boldsymbol{\Phi}^\top \left(\phi(\boldsymbol{y}) - \frac{1}{N}\boldsymbol{\Phi}\mathbf{1}_N \right). \end{aligned} \tag{2.89}$$

The second equality holds, because $\boldsymbol{R}$ is obtained from the singular vectors of the centered gram matrix, and thus $\mathbf{1}_N$ lies in the null space of $\boldsymbol{R}^\top$.

To avoid clutter, let us introduce an abuse of notation in which κ can be applied matrix-wise. Given two matrices $\boldsymbol{Y}_1 \in \mathbb{R}^{d \times N_1}, \boldsymbol{Y}_2 \in \mathbb{R}^{d \times N_2}$, the element in ith row and jth column of $\kappa(\boldsymbol{Y}_1, \boldsymbol{Y}_2) \in \mathbb{R}^{N_1 \times N_2}$ is then given by the κ value of ith column of $\boldsymbol{Y}_1$ and jth column of $\boldsymbol{Y}_2$. Then, we can rewrite Eq. (2.89) as

$$\boldsymbol{F}^\top(\phi(\boldsymbol{y}) + \boldsymbol{b}) = \boldsymbol{R}^\top \left(\kappa(\boldsymbol{Y}, \boldsymbol{y}) - \frac{1}{N}\kappa(\boldsymbol{Y}, \boldsymbol{Y})\mathbf{1}_N \right). \tag{2.90}$$

We eliminated any mentioning of ϕ or $\boldsymbol{\Phi}$ in Eq. (2.90) by replacing it with κ. Thus, all it takes to compute the KPCA parameterization of a data point $\boldsymbol{y} \in \mathbb{R}^d$ are the matrices $\boldsymbol{R}$ and $\boldsymbol{Y}$.

The matrix $\boldsymbol{R}$ can be computed from

$$\begin{aligned} \boldsymbol{K}^{\mathrm{c}} =& \boldsymbol{\Phi}^{\mathrm{c}\top} \boldsymbol{\Phi}^{\mathrm{c}} \\ =& \left(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}_N^\top \right) \boldsymbol{\Phi}^\top \boldsymbol{\Phi} \left(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}_N^\top \right) \\ =& \left(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}_N^\top \right) \kappa(\boldsymbol{Y}, \boldsymbol{Y}) \left(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}_N^\top \right), \end{aligned} \tag{2.91}$$

implying that we do not need $\boldsymbol{\Phi}$ or ϕ for computing $\boldsymbol{R}$, either. The procedure is summarized in Algorithm 5. Once $\boldsymbol{R}$ has been obtained by means of Algorithm 5, the kernel PCA parameterization of a data point $\boldsymbol{y}$ can be computed by means of

Eq. (2.90). For the training data $\boldsymbol{Y}$ itself, the kernel PCA representation can be written as

$$\begin{aligned}\boldsymbol{R}^\top \left(\kappa(\boldsymbol{Y},\boldsymbol{Y}) - \frac{1}{N}\kappa(\boldsymbol{Y},\boldsymbol{Y})\mathbf{1}_N\right) =& \boldsymbol{R}^\top \kappa(\boldsymbol{Y},\boldsymbol{Y}) \left(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}_N^\top\right) \\ =& \boldsymbol{\Lambda}_{(n)}^{(n)^{1/2}} \boldsymbol{U}_{(n)}{}^\top .\end{aligned} \tag{2.92}$$

2.6. Deep Learning

With the increase in data and hardware resources, *deep learning*, i.e., the field that studies neural networks with a considerable number of layers has become somewhat synonymous to all research involving neural nets. Meanwhile, the traditional definition of a neural network as an alternating composition of learnable affine and fixed element-wise functions does no longer do justice to the deep learning reality that has come up with techniques such as pooling [106], dropout [115], batch normalization [57], skip connections [54], and so forth. It would thus go way beyond the scope to have a comprehensive discussion of how neural networks can be defined and implemented, as other resources have done a better job on this, e.g. [46]. Instead, let us refrain to the following characterization of neural networks that should be sufficient for our aims. A neural network is a function

$$f_{\boldsymbol{\theta}} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2} \tag{2.93}$$

that is parameterized by its trainable weights and biases described by $\boldsymbol{\theta} \in \mathbb{R}^D$. Neural nets are usually trained by some variation of gradient descent.

Apart from providing a strong inductive bias for visual data [82, 121] the success of deep neural nets is to a large part due to their adaptability to large and diverse data sets. This is made possible by two aspects of the training process. First, neural networks are typically compositions of simple functions, such that the gradient can be easily computed by means of the chain rule (back-propagation). Second, the loss is defined as an expected value over the data distribution $p(\mathbf{y})$. That is to say, given a a differentiable, scalar function $L\colon \mathbb{R}^{d_2} \to \mathbb{R}$, the loss can be written as

$$L_{\mathbb{E}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{y}\sim p(\mathbf{y})}\left[L(f_\theta(\mathbf{y}))\right]. \tag{2.94}$$

In practice, the expectation is estimated via the sum over the samples. The loss can be then optimized via *stochastic learning* [21], by computing unbiased estimators of the gradient from a subset of the training samples.

For our task of generalizing the affine mapping described by $\boldsymbol{C}, \bar{\boldsymbol{y}}$ in Eq. (1.5b) to a non-linear function, it would be convenient to have a neural network that maps from the latent representation in $\mathbb{R}^n$ to the observation space $\mathbb{M}$. Classically, this can be achieved using an *autoencoder* [68]. The term refers to a neural network that is trained to copy its input to the output and producing a latent representation in the process. (Deep) autoencoders can be seen as a generalization of PCA that results from stripping PCA of the linearity assumption and replacing the affine representation and reconstruction functions in Section 2.1 by two neural networks. Let us denote the functions implemented by the neural networks as $\Gamma^{+}_{\boldsymbol{\vartheta}} : \mathbb{R}^d \to \mathbb{R}^n$ that maps high-dimensional data samples to a low-dimensional latent space, and $\Gamma_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^d$ that serves as the respective reconstruction mapping. Autoencoders are usually trained by minimizing some form of reconstruction loss L_{AE}, similar to Eq. (2.77). Given a training data set $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$, this yields the optimization problem

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^D, \boldsymbol{\vartheta}\in\mathbb{R}^{\tilde{D}}}{\arg\min} \sum_{i=1}^{N} L_{\mathrm{AE}}(\Gamma_{\boldsymbol{\theta}}(\Gamma^{+}_{\boldsymbol{\vartheta}}(\boldsymbol{y}_i)), \boldsymbol{y}_i). \tag{2.95}$$

Let us assume the loss L_{AE} in Eq. (2.95) to be the squared ℓ_2 error, for simplicity.

Ideally, if $\mathbb{M}$ is our observation space, the autoencoder learns a function $\Gamma_{\boldsymbol{\theta}}$ such that its restriction to $\mathbb{M}$ is injective with $\Gamma_{\boldsymbol{\theta}}^{-1} = \Gamma^{+}_{\boldsymbol{\vartheta}}$. That would make $\Gamma_{\boldsymbol{\theta}}$ an appropriate choice for the observation function in Eq. (1.6b). If we follow the same approach as in Section 2.1, i.e., framing the task as the prediction problem

$$\min_{\boldsymbol{\theta}, \boldsymbol{\vartheta}, \boldsymbol{A}} \sum_{t=1}^{N-1} \|\boldsymbol{y}_{t+1} - \Gamma_{\boldsymbol{\theta}}(\boldsymbol{A}\Gamma^{+}_{\boldsymbol{\vartheta}}(\boldsymbol{y}_t))\|^2, \tag{2.96}$$

then we can go about solving the problem by minimizing for all three parameters by stochastic gradient descent. Unlike to Section 2.1, however, it may not be a great idea to learn the observation function separetely from the state transition matrix. The reason is that in the linear case, this step was justified by Eq. (2.22) which states that for natural images, the approximation $\boldsymbol{C}^\top(\boldsymbol{Y}^{\mathrm{c}})_+ \approx \boldsymbol{C}^\top(\boldsymbol{Y}^{\mathrm{c}})_-$ holds. This may not be true for this non-linear case as $\mathbb{M}$ is in general not a blurred-out model for the video frames. The difficulty of learning a visual process model using deep networks is thus not as much about inferring the parameters $\Gamma_{\boldsymbol{\theta}}, \Gamma^{+}_{\boldsymbol{\vartheta}}, \boldsymbol{A}$, but rather doing so jointly.

Deep Generative Models

An alternative to autoencoders for learning the model in Eq. (1.6) are deep generative models. Many of these models, notably the *variational autoencoder* (VAE) [66, 37] and the *generative adversarial net* (GAN) [47], aim at learning an architecture that maps samples from a standard Gaussian distribution to samples that are distributed in the same way as the training data. This is a helpful prerequisite for the model in Eq. (1.6), as the latent states $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are zero-mean Gaussian-distributed.

Larning a neural network $\Gamma_{\boldsymbol{\theta}}$ that maps a random variable $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; \mathbf{0}, \boldsymbol{I}_n)$ to a random variable $\mathbf{y} \sim p(\mathbf{y})$, where $p(\mathbf{y})$ denotes some data distribution, is challenging. One naive approach would be a Monte Carlo based method, in which we sample $\tilde{N}$ samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\tilde{N}} \in \mathbb{R}^n$ from i.i.d. standard Gaussian noise, transform them via $\Gamma_{\boldsymbol{\theta}}$, and build a *kernel density estimator* (KDE) [93] model from the transformed samples, e.g. with isotropic gaussian windows of bandwidth $\sigma_{\mathbf{y}}$. Then, we could formulate the log-likelihood objective

$$L_{\mathrm{MC}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \left(\frac{1}{N\sigma^d\sqrt{(2\pi)^d}} + \sum_{j=1}^{\tilde{N}} \exp\left(\frac{-\|\boldsymbol{y}_i - \Gamma_{\boldsymbol{\theta}}(\boldsymbol{x}_j)\|^2}{2\sigma^2} \right) \right), \tag{2.97}$$

where $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N \in \mathbb{R}^d$ is the training set. Obviously, the quality of the model will depend on the number $\tilde{N}$ of Monte Carlo samples. However, it is impossible to write the loss as a plain sum of both the training and Monte Carlo samples, which makes it infeasible to apply stochastic learning over $\boldsymbol{x}_1 \ldots \boldsymbol{x}_{\tilde{N}}$. Thus, realistically, such a model could only be trained for a very small choice of $\tilde{N}$ which drastically limits its quality. Therefore, more sophisticated approaches to this problem need to be considered, which is the subject of the remaining sections in this chapter.

Variational Autoencoders

Consider a latent variable model, described by the real-valued and finite-dimensional parameter $\boldsymbol{\theta}$, in which the observable data variable $\mathbf{y} \sim p_{\boldsymbol{\theta}}(\mathbf{y})$ is related to the latent variable $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; 0, \boldsymbol{I}_n)$ through the conditional probability density $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$. We assume, that given an observation of $\mathbf{x}$, we can sample from $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ by applying some transformation that is parameterized by $\boldsymbol{\theta}$. For a set of independently drawn observations $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$, the log-likelihood with respect to $\boldsymbol{\theta}$ can be written as

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_1 \ldots, \boldsymbol{y}_N) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i). \tag{2.98}$$

By applying the Bayes rule, the summands in Eq. (2.98) can be rewritten in terms of the posterior density $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{y})$ as follows.

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i) &= \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}_i)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{y}_i)} \\ &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}_i) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{y}_i). \end{aligned} \tag{2.99}$$

Additionally, since $\boldsymbol{y}_i$ is not a random variable, applying the expected value to $\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i)$ does not affect it, so that we can write

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i) =& \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i)] \\ =& \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)}[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{y}_i) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)]. \end{aligned} \tag{2.100}$$

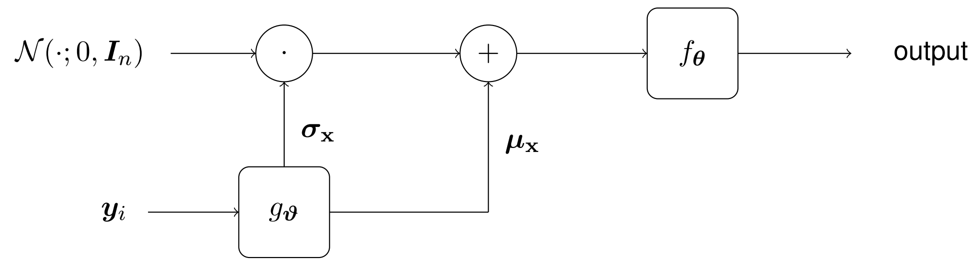

$\mathcal{N}(\cdot;0,\boldsymbol{I}_n)$
$\cdot$
$+$
$f_{\boldsymbol{\theta}}$
output
$\boldsymbol{\sigma}_{\mathbf{x}}$
$\boldsymbol{\mu}_{\mathbf{x}}$
$\boldsymbol{y}_i$
$g_{\boldsymbol{\vartheta}}$

Figure 2.2.: VAE architecture during training. Once $\boldsymbol{\theta}$ is learned, the data distribution can be reproduced by sampling from i.i.d. standard Gaussian noise, and transforming the noise samples via $f_{\boldsymbol{\theta}}$.

Finally, we can decompose $p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{y}_i)$ once again using the Bayes rule, and rewrite the term as

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i) =& \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\mathbf{x}) + \log p(\mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)] \\ =& \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\mathbf{x})] - D_{\mathrm{KL}}\left(p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)\|p(\mathbf{x})\right).\end{aligned} \tag{2.101}$$

The advantage of Eq. (2.101) is that by sampling from $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)$, an unbiased estimator of $\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i)$ can be obtained, which is crucial to performing stochastic optimization. Unfortunately, $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)$ is typically not known. VAEs overcome this problem via replacing it by a parameterized approximation, denoted by $q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)$. This yields

$$\mathbb{E}_{\mathbf{x}\sim q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\mathbf{x})] - D_{\mathrm{KL}}(q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)\|p(\mathbf{x})). \tag{2.102}$$

Maximizing the term in Eq. (2.102) leads to the approximate maximization of the log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i)$, if $q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)$ is similar to $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)$. This can be enforced by maximizing Eq. (2.102) not only with respect to $\boldsymbol{\theta}$, but also with respect to $\boldsymbol{\vartheta}$. The reason is that Eq. (2.102) is equal to the *variational lower bound*, i.e.,

$$\begin{aligned}&\mathbb{E}_{\mathbf{x}\sim q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\mathbf{x})] - D_{\mathrm{KL}}(q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)\|p(\mathbf{x})) \\ =& \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i) - D_{\mathrm{KL}}\left(q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i))\|p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)\right),\end{aligned} \tag{2.103}$$

and therefore, maximizing Eq. (2.102) w.r.t. $\boldsymbol{\vartheta}$ minimizes the KLD between the posterior density $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{y}_i)$ and its parameterized approximation $q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)$.

Fig. 2.2 depicts an overview of the VAE architecture. The VAE implements the conditional distributions $q_{\boldsymbol{\vartheta}}(\mathbf{x}|\boldsymbol{y}_i)$ and $p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\mathbf{x})$ by means of the encoder and the decoder of an auto-encoder architecture, respectively. The specific definitions of these conditional distributions are motivated by practical considerations. For $p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\mathbf{x})$, we assume the model

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{x}|f_{\boldsymbol{\theta}}(\mathbf{x}), \sigma_{\mathbf{y}}^2\boldsymbol{I}_d), \tag{2.104}$$

where $f_{\boldsymbol{\theta}}(\mathrm{x})$ is implemented by the decoder with the weights $\boldsymbol{\theta}$, and $\sigma_{\mathbf{y}}^2$ is the additive noise variance to be chosen by hand. The distribution $q_{\boldsymbol{\vartheta}}(\mathrm{x}|\boldsymbol{y}_i)$ is modeled as a covariance-free Gaussian distribution

$$q_{\boldsymbol{\vartheta}}(\mathrm{x}|\boldsymbol{y}_i) = \mathcal{N}(\mathrm{x}; \boldsymbol{\mu}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta}), \operatorname{diag}(\boldsymbol{\sigma}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta}))^2), \tag{2.105}$$

where the mean $\boldsymbol{\mu}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta})$ and the deviation vector $\boldsymbol{\sigma}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta})$ is computed by applying the encoder $g_{\boldsymbol{\vartheta}}$ with weights $\boldsymbol{\vartheta}$ to the training samples, i.e.

$$\begin{aligned} \boldsymbol{\mu}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta}) &= \begin{bmatrix} \boldsymbol{I}_n & \mathbf{0} \end{bmatrix} g_{\boldsymbol{\vartheta}}(\boldsymbol{y}_i) \ \in \mathbb{R}^n, \\ \boldsymbol{\sigma}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta}) &= \begin{bmatrix} \mathbf{0} & \boldsymbol{I}_n \end{bmatrix} g_{\boldsymbol{\vartheta}}(\boldsymbol{y}_i) \ \in \mathbb{R}^n. \end{aligned} \tag{2.106}$$

That way, by multiplying the variational lower bound in Eq. (2.102) by (-2) and removing any additive terms not depending on $\boldsymbol{\theta}, \boldsymbol{\vartheta}$, we can formulate the VAE loss as

$$\begin{aligned} L_{\mathrm{VAE}}(\boldsymbol{\theta}, \boldsymbol{\vartheta}) = \sum_{i=1}^{N} \frac{1}{\sigma_y^2} \mathbb{E}_{\mathrm{x} \sim q_{\boldsymbol{\vartheta}}(\mathrm{x}|\boldsymbol{y}_i)} \left[\|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\mathrm{x})\|^2 \right] + \|\boldsymbol{\mu}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta})\|^2 \\ + \mathbf{1}_n^\top (\boldsymbol{\sigma}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta}) - \log \boldsymbol{\sigma}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta})), \end{aligned} \tag{2.107}$$

and optimize it via stochastic gradient decent. For each iteration, this is done by drawing a sample $\boldsymbol{y}_i$ from the training data to compute an estimator of the sum, and subsequently drawing one or several samples from the conditional distribution $q_{\boldsymbol{\vartheta}}(\mathrm{x}|\boldsymbol{y}_i)$ to compute an estimator of the expected value in Eq. (2.107). The latter is carried out by generating white Gaussian noise, and *reparameterizing* it by means of the encoder outputs, $\boldsymbol{\mu}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta}), \boldsymbol{\sigma}_{\mathrm{x}}(\boldsymbol{y}_i, \boldsymbol{\vartheta})$.

After successful training, i.e., when the VAE succeeded in modeling the training data distribution, samples from $p_{\boldsymbol{\theta}}(\mathrm{y})$ can be created by generating white Gaussian noise, applying $f_{\boldsymbol{\theta}}$ to it, and adding zero-mean Gaussian noise with variance $\boldsymbol{\sigma}_{\mathbf{y}}$ to the decoder output. This last perturbation step is a consequence of the model assumption in Eq. (2.104) which requires that the mapping from x to y is perturbed by additive noise. This is necessary for the loss provided in Eq. (2.107) to be well-defined.

Generative Adversarial Networks

A GAN consists of two neural networks, namely a *discriminator* network, denoted by $D_{\boldsymbol{\vartheta}}$ in the following, and a *generator* network, referred to as $G_{\boldsymbol{\theta}}$. During training, the generator takes samples of standard Gaussian noise in $\mathbb{R}^n$ and transforms them into samples in $\mathbb{R}^d$. The discriminator evaluates the resulting samples in $\mathbb{R}^d$ and returns a score that indicates how likely the sample was taken from the training set as opposed to being generated by $G_{\boldsymbol{\theta}}$. By contrast, the generator aims at producing samples that are indistinguishable from training data for the discriminator. To formalize this idea as an optimization problem, the discriminator output is interpreted as the probability

$$D_{\boldsymbol{\vartheta}}(\mathbf{y}) = P(\boldsymbol{y} \text{ is real}) = 1 - P(\boldsymbol{y} \text{ is generated}). \tag{2.108}$$

The training of $\boldsymbol{\theta}$ should be thus carried out in a way such that the likelihood expression

$$\mathbb{E}_{\mathbf{y}\sim p(\mathbf{y})}[\log D_{\boldsymbol{\vartheta}}(\mathbf{y})] + \mathbb{E}_{\mathbf{x}\sim N(\mathbf{x};0,\boldsymbol{I}_k)}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(\mathbf{x})))] \tag{2.109}$$

is maximized. Likewise, the point of the generator is to produce samples from Gaussian noise that appear real to the discriminator, implying that $\boldsymbol{\vartheta}$ should be trained to minimize

$$\mathbb{E}_{\mathbf{x}\sim N(\mathbf{x};0,\boldsymbol{I}_k)}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(\mathbf{x})))]. \tag{2.110}$$

Together, these two aims yield the objective

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\vartheta}} \mathbb{E}_{\mathbf{y}\sim p(\mathbf{y})}[\log D_{\boldsymbol{\vartheta}}(\mathbf{y})] + \mathbb{E}_{\mathbf{x}\sim N(\mathbf{x};0,\boldsymbol{I}_k)}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(\mathbf{x})))], \tag{2.111}$$

as it was formulated originally [47].

The objective in Eq. (2.111) is optimized by alternating between the maximization for $\boldsymbol{\vartheta}$ and minimization for $\boldsymbol{\theta}$. To this end, samples from the training set and the standard Gaussian distribution are drawn for estimating the respective expectations.

Traditionally, Eq. (2.111) and similar objectives are interpreted as incentives to minimize a divergence measure between the data and the generated distribution [47]. Let us denote these distributions by p_D and p_G, respectively. The output of an ideal discriminator $\hat{D}$ that, for a sample $\boldsymbol{y}$ returns the probability that it belongs to the data distribution and not the generated distribution, can be computed as

$$\hat{D}(\boldsymbol{y}) = \frac{p_D(\boldsymbol{y})}{p_D(\boldsymbol{y}) + p_G(\boldsymbol{y})}. \tag{2.112}$$

Substituting this into Eq. (2.111) yields

$$\begin{aligned} &\mathbb{E}_{\mathbf{y}\sim p_D(\mathbf{y})}[\log \hat{D}(\mathbf{y})] + \mathbb{E}_{\mathbf{y}\sim p_G(\mathbf{y})}[\log(1 - \hat{D}(\mathbf{y})))] \\ =&D_{\mathrm{KL}}\left(p_D \middle\| \frac{p_D + p_G}{2}\right) + D_{\mathrm{KL}}\left(p_G \middle\| \frac{p_D + p_G}{2}\right) - \log(4). \end{aligned} \tag{2.113}$$

The non-constant part $D_{\mathrm{KL}}\left(p_D \| (p_D + p_G)/2\right) + D_{\mathrm{KL}}\left(p_G \| (p_D + p_G)/2\right)$ is called the *Jensen-Shannon* divergence and is a measure of dissimilarity between p_D and p_G that the generator aims to minimize.

3. Kernelized Alignment Distances for Streams of Histograms

A classical problem in visual process research is the classification of observed video sequences. Deep learning based end-to-end approaches, e.g. [9, 5], typically achieve state-of-the art performance. Still, they are not always the method of choice for visual processes, since it may be harder to gather sufficient amounts of data to train these models, compared to the case of still-image classification. Also, such approaches cannot be employed in recognition tasks other than classification, where no prior labeling is given, such as clustering or retrieval.

Many classification frameworks for visual processes have thus refrained to a more traditional, feature-based approach, in which the video sequences are first processed by a *feature extraction* (FE) procedure that computes expressive descriptors of the sequences. Subsequently, these features are then subjected to a dissimilarity measure that can be applied, for instance in the context of k-NN or NCC classification. Classically, this can be done by defining a distance measure on computed LDS models [105, 25] or using multi-scale spatio-temporal decompositions [41, 59] or sparse representations [98]. Furthermore, a considerable number of approaches collects features designed for 2D images from a video, by applying them in *three orthogonal planes* (TOP) of the video cuboid [133, 56, 10, 88].

A considerable share of the works focussing on dynamic scenes or high-resolution dynamic textures, such as [86, 87], are based on distribution models of localized or scale-dependent spatio-temporal descriptors calculated throughout the video in question. The motivation is to recognize a video sequence as a "sum of its parts". A video that exhibits many features describing cars at several spatial and temporal locations can be then easily recognized as a traffic scene, for example. The computed features typically capture some appearance and dynamics properties. Many of the aforementioned methods compute spatially and temporally localized 3D features and a spatio-temporally global distribution thereof. A noteworthy example is the *Bag of Systems* (BoS) [100] that divides the video into small spatio-temporal cuboids and computes an LDS model from each cuboid. This permits to model the visual process as a distribution of LDS's. The approach discussed in the following is also based on distributions of localized features. However, unlike the methods mentioned above, it follows the paradigm of Eq. (1.6) that assumes a semi-linear state space mechanism as a generating model of the visual process. Distribution based spatial features are computed for each video frame individually, while pre-

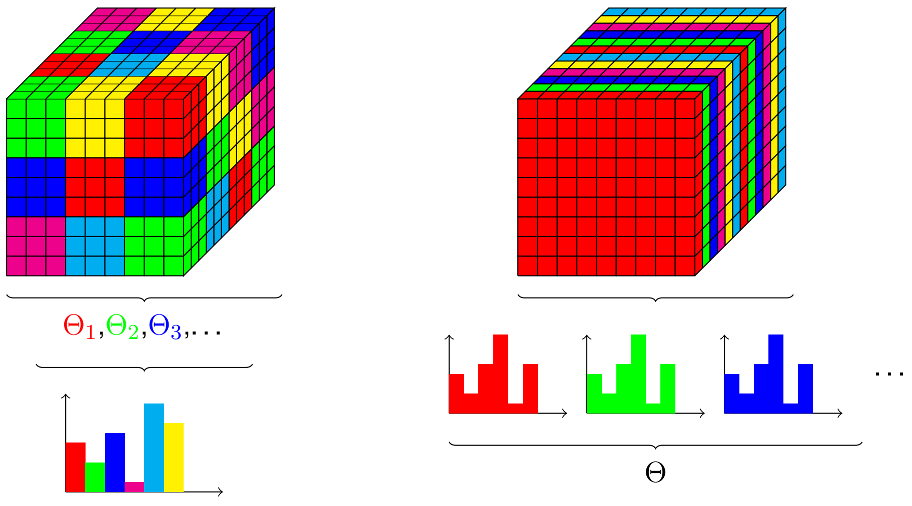


Figure 3.1.: Bag of Systems vs. System of Bags: While a BoS describes the video as a global distribution of spatially and temporally localized systems $\Theta_1, \Theta_2, \ldots,$ a SoB describes the video as a temporally localized but spatially global distribution of features, that changes over time according to one single system Θ.

serving their temporal order. The resulting model is referred to as a *System of Bags* (SoB) in the following, as it consists of a system that models the temporal evolution of distributions (bags). Figure 3.1 illustrates the difference between BoS and SoB.

One may assume that ditching 3D spatio-temporal features in favor of purely spatial ones make the descriptors less distinguishable, but this depends on the type of visual process in question. For example, when it comes to dynamic scenes, our everyday experience indicates that we can tell them apart by observing isolated frames. That means that it can make sense to rely on FE methods for still images which have become impressively efficient over the course of the last two decades, and integrate them into a temporal model to provide complementary information that cannot be captured from one single frame. This information should tell us something about how the image frames of a visual process evolve as a whole, e.g. due to changing weather conditions in an outdoor scene, rather than about localized movements that are semantically irrelevant.

The term *bag* refers to the *Bag of Words* (BoW) model that has originated in document classification [52] and has since been generalized to the field of computer vision [42]. Simply put, the idea is to collect localized features, e.g. via the SIFT algorithm [77] from a set of images, construct a so-called *Codebook* of these features, and build image descriptors by means of histograms that indicate how often the Codebook features are contained in the image at hand. BoWs are not the only type of histogram descriptors in still image processing. Textures, for instance, can be well described by *local binary patterns* (LBP) [90]. Using these types of descriptors as the observation vectors in a visual process requires to learn a model of the temporal evolution of histograms. Since sets of histograms cannot be well modeled by subspaces of $\mathbb{R}^d$ due to their norm constraints, this is done in a kernel feature space in what follows. The dynamic model employed for it is the *Kernelized LDS* [26]. Once a KLDS model has been learned for each video of a data set, the visual processes can be classified by means of the extracted KLDS parameters. Modeling streams of histograms by a KLDS in order to perform classification has been first proposed in [28]. The contribution of this chapter is thus not the introduction of this type of model itself. Rather, the following sections propose approaches to efficiently adapt and apply this type of model to the problem of dynamic scene and dynamic texture classification. It puts its emphasis on employing the Alignment distance for classification purposes, once the KLDS features have been computed. More specifically, the contributions of this chapter are the following.

- Since the Alignment distance has previously not been applied in the context of KLDS's, a kernelized version of the Alignment distance is derived. Additionally, rather than using the original definition from [2], the Alignment distance is used as it is defined in Section 2.3, allowing for better capturing the static aspects of the visual processes.
- For NCC classification, *Fréchet* means of KLDS's are computed. The method is based on [2], but the computation is done in the kernel feature space. This requires some adaptations. Notably, kernelization causes issues with scalability which are mitigated by an approach inspired by *Nyström* interpolation [40].
- Some important properties of the Alignment distance are revisited. Specifically, it is explained why the square root of the Alignment distance is a metric on the quotient space $\mathcal{QO}_d^n$ of statistically equivalent LDS's as well as on its kernelized counterpart. This has interesting implications on how to theoretically justify NCC classifications using Fréchet means. This claim was made before in [3], but, to the author's best knowledge, never explicitly derived.
- Unlike [28] that discuses human action recognition, the work conducted in this chapter focuses on dynamic textures and dynamic scenes in the experimental section. The type of features collected from the videos need to be different from the optical flow based descriptors used in [28]. Particularly, dynamic scenes require a richer type of descriptors. For dynamic scenes, bags of visual features extracted via the SURF algorithm [14] are employed. Dynamic textures are described by means of LBPs. [124].

Experimentally, these contributions are validated by two observations. On the one hand, it is shown that, given the KLDS parameters describing a set of SoBs, the kernelized version of the Alignment distance yields better classification results than the common choices for divergence measures on KLDS's. Beyond that, it is demonstrated that by choosing the right histogram features for the video frames of a visual process, an SoB can outperform comparable, non-deep learning state-of-the-art methods of k-NN and NCC classification of dynamic textures and dynamic scenes.

3.1. Systems of Bags

Often, the observations $\boldsymbol{y}_1, \boldsymbol{y}_2, \dots \in \mathbb{R}^d$ produced by a visual process are RGB or grayscale image frames with d_1 rows, d_2 columns and c color channels that have been flattened to vectors of dimension $d = d_1 d_2 c$. However, this does not always have to be the case. If temporally localized dynamics are irrelevant for the recognition of a visual process, then it may be more sensible to convert the image frames to feature representations that are known to perform well under distance based recognition tasks. That may cause some loss of certain irrelevant information such as small-scale movements of objects in the video, but emphasizes important information such as the kinds of objects that are present in the video.

For many successful still images classification approaches, the employed FE technique results in a set of histograms [49, 31, 90, 42]. In this chapter, it will be assumed that the matrix

$$\boldsymbol{Y} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_N] \in \mathbb{R}^{d \times N} \tag{3.1}$$

contains N histograms obtained this way from a video sequence. Each one of these histograms should be a good descriptor of the video frame from which it was computed. In order to combine all these histograms individually as well as their temporal evolution into a strong descriptor of the overall video, a model is required that could explain how the columns of $\boldsymbol{Y}$ came to be.

Kernelized Linear Dynamic Systems

A histogram is a vector in $\mathbb{R}^d$ with non-negative entries that sum up to one. Let us use the notation $(\cdot)_i$ for ith element of a vector. The observation vectors of the visual process at hand are thus constrained to the set

$$\mathbb{M} = \{\boldsymbol{y} \in \mathbb{R}^d \mid (\boldsymbol{y})_i \geq 0 \; \forall i \in \{1, \dots, d\}, \; \|\boldsymbol{y}\|_1 = 1\}. \tag{3.2}$$

It is not promising to model trajectories in $\mathbb{M}$ by an LDS, simply because it looks nothing like an affine subspace of $\mathbb{R}^d$. However, kernels have come a long way in machine learning problems involving histograms and other distribution based descriptors [104, 94, 12, 60]. Some argue that kernels achieve their performance by *linearizing* the data in the kernel feature space [82]. If this is the case, then we can hope to find a proper kernel that allows us to map $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ to an appropriate feature space, where the affine subspace assumption is no longer out of question.

Algorithm 6 : Kernelized linear dynamic system

```
Input : Video Sequence $\boldsymbol{Y} \in \mathbb{R}^{d \times N}$, target dimension $n \in \mathbb{N}$
1 $\boldsymbol{K} \leftarrow \kappa(\boldsymbol{Y}, \boldsymbol{Y})$;                          // Computing Gram matrix
2 $\boldsymbol{K}_{\mathrm{c}} \leftarrow (\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top)\boldsymbol{K}(\boldsymbol{I}_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top)$;        // Centering data
3 $\boldsymbol{U}, \boldsymbol{\Lambda}, \boldsymbol{U} \leftarrow \mathrm{SVD}(\boldsymbol{K}_{\mathrm{c}})$;
4 $\boldsymbol{R} \leftarrow \boldsymbol{U}_{(n)} {\boldsymbol{\Lambda}_{(n)}^{(n)}}^{-1/2}$;
5 $\begin{bmatrix}\boldsymbol{x}_1 & \cdots & \boldsymbol{x}_N\end{bmatrix} \leftarrow {\boldsymbol{\Lambda}_{(n)}^{(n)}}^{1/2} \boldsymbol{U}_{(n)}^\top$;          // Latent space states
6 $\boldsymbol{A} \leftarrow \arg\min_{\tilde{\boldsymbol{A}} \text{ s.t. } \|\tilde{\boldsymbol{A}}\|_2 < 1} \sum_{i=1}^{N-1} \|\boldsymbol{x}_{t+1} - \tilde{\boldsymbol{A}}\boldsymbol{x}_t\|^2$ ;            // State
  transition matrix
7 $\boldsymbol{B} \leftarrow \frac{1}{\sqrt{N-1}}(\sum_{i=1}^{N-1}(\boldsymbol{x}_{t+1} - \boldsymbol{A}\boldsymbol{x}_t)(\boldsymbol{x}_{t+1} - \boldsymbol{A}\boldsymbol{x}_t)^\top)^{\frac{1}{2}}$;            // Noise
  covariance transform
8 $\boldsymbol{\beta} \leftarrow \frac{1}{N}\mathbf{1}_N$;                           // Needed to express $\bar{\boldsymbol{\varphi}}$
Output : LDS parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{R}, \boldsymbol{\beta}, \boldsymbol{Y}$
```

Consider the semi-linear state space model of a visual process that was introduced first in Eq. (1.6). Recall that Eq. (1.6a) describes a VAR noise model. Let $\mathbb{H}$ be a separable Hilbert space and the function $\phi : \mathbb{R}^d \to \mathbb{H}$ a mapping from $\mathbb{R}^d$ to said space. Furthermore, let $\bar{\boldsymbol{\varphi}} \in \mathbb{H}, \boldsymbol{\Phi} \in \mathbb{H}^n$ be defined. Let us also make the simplifying assumption that the restriction of ϕ to $\mathbb{M}$ is bijective. Although this assumption does not generally hold, it is a justifiable approximation for practically relevant observation spaces and appropriately chosen kernels [85]. By making this assumption, we can express the observation function Γ as

$$\begin{aligned} \Gamma : \mathbb{R}^n &\to \mathbb{R}^d, \\ \boldsymbol{x} &\mapsto \phi^{-1}(\bar{\boldsymbol{\varphi}} + \boldsymbol{\Phi}\boldsymbol{x}), \end{aligned} \tag{3.3}$$

where the matrix-vector multiplication is understood in the sense of Section 2.5. If we do not have an explicit representation of ϕ, but can only describe it implicitly by means of a kernel κ, we call the model in Eq. (3.3) a *kernelized linear dynamic system* (KLDS). In other words, a KLDS follows a linear dynamic in $\mathbb{H}$ described by

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{v}_t, \tag{3.4a}$$

$$\boldsymbol{h}_t = \bar{\boldsymbol{\varphi}} + \boldsymbol{\Phi}\boldsymbol{x}_t, \tag{3.4b}$$

such that $\boldsymbol{h}_t = \phi(\boldsymbol{y}_t)$ holds for the observations $\boldsymbol{y}_1, \boldsymbol{y}_2 \ldots$ This concept was first proposed in [26], along with an algorithm based on kernel PCA to extract the parameters describing a KLDS from a video $\boldsymbol{Y}$ of a visual process. Algorithm 6 summarizes that procedure. Note the difference to Algorithm 2 with regards to the output. Unlike Algorithm 6, Algorithm 2 returns $\boldsymbol{C}$ and $\bar{\boldsymbol{y}}$, along with $\boldsymbol{A}, \boldsymbol{B}$. Now, the corresponding parameters $\boldsymbol{\Phi}, \bar{\boldsymbol{\varphi}}$ live in Hilbert spaces which we cannot describe in an explicit manner. This is the reason why these parameters cannot be directly returned by the algorithm. However, any inner product expression containing these two parameters can be represented by means of κ and the parameters $\boldsymbol{R}, \boldsymbol{\beta}$ and $\boldsymbol{Y}$ which are thus returned instead. This also means that we have no way to directly compute the observation function Γ, making it impossible to carry out video synthesis in such a straightforward manner as it is done in the linear case in Section 2.2.

KLDS features

Taking into account the state space basis invariance we have discussed in the previous chapter, we can introduce the KLDS set $\mathcal{K}_d^n$ and an equivalence relation $\mathcal{R}_d^n$, in analogy to $\mathcal{O}_d^n$ in Eq. (2.35) and $\mathcal{Q}_d^n$ (Eq. (2.36)) in Section 2.1. Let us define $\mathcal{K}_d^n$ as

$$\begin{aligned}\mathcal{K}_d^n = \{(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{R}, \boldsymbol{\beta}, \boldsymbol{Y}) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}^{N\times n} \times \mathbb{R}^{N} \times \mathbb{R}^{d\times N} \\ \mid N \in \mathbb{N},\ \|\boldsymbol{A}\|_2 < 1,\ \boldsymbol{R}^\top \kappa(\boldsymbol{Y}, \boldsymbol{Y})\boldsymbol{R} = \boldsymbol{I}_n\}.\end{aligned} \tag{3.5}$$

The equality $\boldsymbol{R}^\top \kappa(\boldsymbol{Y}, \boldsymbol{Y})\boldsymbol{R} = \boldsymbol{I}_n$ states that $\mathcal{K}_d^n$ contains KLDS's for which the feature space observation matrix $\boldsymbol{\Phi} \in \mathbb{H}^n$ has orthonormal columns. As the next step, let us introduce the notation

$$\boldsymbol{Q} \bullet (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{R}, \boldsymbol{\beta}, \boldsymbol{Y}) = (\boldsymbol{Q}\boldsymbol{A}\boldsymbol{Q}^\top, \boldsymbol{Q}\boldsymbol{B}, \boldsymbol{R}\boldsymbol{Q}^\top, \boldsymbol{\beta}, \boldsymbol{Y}), \tag{3.6}$$

for $\boldsymbol{Q} \in \mathrm{O}(n)$. The according equivalence relation that, in analogy to Eq. (2.36), denotes all pairs of KLDS's with equivalent statistical properties is given by

$$\mathcal{R}_d^n = \{(\Xi_1, \Xi_2) \in \mathcal{K}_d^n \times \mathcal{K}_d^n \mid \exists \boldsymbol{Q} \in \mathrm{O}(n) \text{ s.t. } \Xi_1 = \boldsymbol{Q} \bullet \Xi_2\}. \tag{3.7}$$

These definitions lead up to the quotient set

$$\mathcal{R}\mathcal{K}_d^n = \{\{\boldsymbol{Q} \bullet \Xi \mid \boldsymbol{Q} \in \mathrm{O}(n)\} \mid \Xi \in \mathcal{K}_d^n\}, \tag{3.8}$$

which generalizes $\mathcal{Q}\mathcal{O}_d^n$ from Chapter 2.

3.2. The Alignment Distance on KLDS's

Both the Martin distance and the Max SV distance on pairs of LDS's can be generalized to KLDS's [26, 28]. However, the dissimilarity of our choice will be the Alignment distance here. The reason is that it permits us to decide how much each system parameter should contribute to the classification process. This is an advantage in the setting at hand, as it may be considerably more important to characterize the set of histograms generated by the visual process than how these histograms develop over time. That is to say, the parameters $\boldsymbol{R}$ and $\boldsymbol{\beta}$ may be more expressive than $\boldsymbol{A}$

or $\boldsymbol{B}$. Unlike the Alignment distance, neither Max SV nor the Martin distance allows for such prioritization, and neglect entirely the $\boldsymbol{\beta}$ parameter.

After defining the Alignment distance for pairs of KLDS's, this section shows how to derive a closed-form expression of it, using the KLDS parameters obtained by Algorithm 6. It proceeds then to discuss its property as a metric on $\mathcal{RK}_d^n$. Computation using a Jacobi algorithm and the convergence behavior of this algorithm are also covered.

Derivation

Consider two KLDS's

$$\Xi_1 = (\boldsymbol{A}_1, \boldsymbol{B}_1, \boldsymbol{R}_1, \boldsymbol{\beta}_1, \boldsymbol{Y}_1),\ \Xi_2 = (\boldsymbol{A}_2, \boldsymbol{B}_2, \boldsymbol{R}_2, \boldsymbol{\beta}_2, \boldsymbol{Y}_2). \tag{3.9}$$

Using the form in Eq. (3.4) and denoting the according kernel feature space terms of Ξ_1, Ξ_2 by $\bar{\varphi}_1, \boldsymbol{\Phi}_1$ and $\bar{\varphi}_2, \boldsymbol{\Phi}_2$, respectively, we can derive the *Alignment distance* for KLDS's by simply replacing the norms in $\mathbb{R}^d$ by their respective $\mathbb{H}$ counterpart. Eq. (2.58) then becomes

$$\begin{aligned} d^2_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) = \min_{\boldsymbol{Q}\in\mathrm{O}(n)} & \|\boldsymbol{\Phi}_1 - \boldsymbol{\Phi}_2\boldsymbol{Q}^\top\|_F^2 \\ & + \lambda_{\boldsymbol{A}}\|\boldsymbol{A}_1 - \boldsymbol{Q}\boldsymbol{A}_2\boldsymbol{Q}^\top\|_F^2 \\ & + \lambda_{\boldsymbol{B}}\|\boldsymbol{B}_1 - \boldsymbol{Q}\boldsymbol{B}_2\|_F^2 \\ & + \lambda_{\bar{\varphi}}\|\bar{\varphi}_1 - \bar{\varphi}_2\|_{\mathbb{H}}^2. \end{aligned} \tag{3.10}$$

Multiplying out norms yields the expression

$$\begin{aligned} & d^2_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) \\ & = \min_{\boldsymbol{Q}\in\mathrm{O}(n)} \|\boldsymbol{\Phi}_1\|_F^2 - 2\operatorname{tr}(\boldsymbol{\Phi}_1^\top\boldsymbol{\Phi}_2\boldsymbol{Q}^\top) + \|\boldsymbol{\Phi}_2\|_F^2 \\ & \qquad + \lambda_{\boldsymbol{A}}\left(\|\boldsymbol{A}_1\|_F^2 - 2\operatorname{tr}\left(\boldsymbol{A}_1^\top\boldsymbol{Q}\boldsymbol{A}_2\boldsymbol{Q}^\top\right) + \|\boldsymbol{A}_2\|_F^2\right) \\ & \qquad + \lambda_{\boldsymbol{B}}\left(\|\boldsymbol{B}_1\|_F^2 - 2\operatorname{tr}(\boldsymbol{B}_1^\top\boldsymbol{Q}\boldsymbol{B}_2) + \|\boldsymbol{B}_2\|_F^2\right) \\ & \qquad + \lambda_{\bar{\varphi}}\left(\|\bar{\varphi}_1\|_{\mathbb{H}}^2 - 2\bar{\varphi}_1^\top\bar{\varphi}_2 + \|\bar{\varphi}_2\|_{\mathbb{H}}^2\right). \end{aligned} \tag{3.11}$$

By defining the two functions

$$\begin{aligned} \tau_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) = & \operatorname{tr}(\boldsymbol{\Phi}_1^\top\boldsymbol{\Phi}_1 + \boldsymbol{\Phi}_2^\top\boldsymbol{\Phi}_2) \\ & + \lambda_{\boldsymbol{A}}(\|\boldsymbol{A}_1\|_F^2 + \|\boldsymbol{A}_2\|_F^2) + \lambda_{\boldsymbol{B}}(\|\boldsymbol{B}_1\|_F^2 + \|\boldsymbol{B}_2\|_F^2) \\ & + \lambda_{\bar{\varphi}}(\bar{\varphi}_1^\top\bar{\varphi}_1 - 2\bar{\varphi}_1^\top\bar{\varphi}_2 + \bar{\varphi}_2^\top\bar{\varphi}_2), \end{aligned} \tag{3.12}$$

and

$$\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \Xi_2) = \operatorname{tr}(\boldsymbol{\Phi}_1^\top\boldsymbol{\Phi}_2) + \lambda_{\boldsymbol{A}}\operatorname{tr}(\boldsymbol{A}_1^\top\boldsymbol{A}_2) + \lambda_{\boldsymbol{B}}\operatorname{tr}(\boldsymbol{B}_1^\top\boldsymbol{B}_2), \tag{3.13}$$

while writing the Frobenius norms in $\mathbb{H}$ as trace products, Eq. (3.11) can be rewritten as

$$\begin{aligned} d^2_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1,\Xi_2) =& \tau_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1,\Xi_2) \\ & - 2\max_{\boldsymbol{Q}\in\mathrm{O}(n)} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1,\boldsymbol{Q}\bullet\Xi_2). \end{aligned} \tag{3.14}$$

Using the kernel trick, the substitutions $\boldsymbol{\Phi}_a^\top\boldsymbol{\Phi}_b = \boldsymbol{R}_a^\top\kappa(\boldsymbol{Y}_a,\boldsymbol{Y}_b)\boldsymbol{R}_b$ and $\bar{\boldsymbol{\varphi}}_a^\top\bar{\boldsymbol{\varphi}}_b = \boldsymbol{\beta}_a^\top\kappa(\boldsymbol{Y}_a,\boldsymbol{Y}_b)\boldsymbol{\beta}_b$ for every $a,b\in\{1,2\}$ can be made. This yields

$$\begin{aligned} \tau_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1,\Xi_2) =& 2n + \lambda_{\boldsymbol{A}}(\|\boldsymbol{A}_1\|_F^2 + \|\boldsymbol{A}_2\|_F^2) \\ & + \lambda_{\boldsymbol{B}}(\|\boldsymbol{B}_1\|_F^2 + \|\boldsymbol{B}_2\|_F^2) + \lambda_{\bar{\varphi}}\boldsymbol{\beta}_1^\top\kappa(\boldsymbol{Y}_1,\boldsymbol{Y}_1)\boldsymbol{\beta}_1 \\ & - 2\lambda_{\bar{\varphi}}\boldsymbol{\beta}_1^\top\kappa(\boldsymbol{Y}_1,\boldsymbol{Y}_2)\boldsymbol{\beta}_2 + \lambda_{\bar{\varphi}}\boldsymbol{\beta}_2^\top\kappa(\boldsymbol{Y}_2,\boldsymbol{Y}_2)\boldsymbol{\beta}_2, \end{aligned} \tag{3.15}$$

and finally,

$$\begin{aligned} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1,\Xi_2) =& \operatorname{tr}(\boldsymbol{R}_1^\top\kappa(\boldsymbol{Y}_1,\boldsymbol{Y}_2)\boldsymbol{R}_2) + \lambda_{\boldsymbol{A}}\operatorname{tr}(\boldsymbol{A}_1^\top\boldsymbol{A}_2) \\ & + \lambda_{\boldsymbol{B}}\operatorname{tr}(\boldsymbol{B}_1^\top\boldsymbol{B}_2). \end{aligned} \tag{3.16}$$

Note that the unknown feature space terms $\boldsymbol{\Phi}_1,\boldsymbol{\Phi}_2,\bar{\boldsymbol{\varphi}}_1,\bar{\boldsymbol{\varphi}}_2$ are absent from Eq. (3.15) and Eq. (3.16). They are replaced by the known expressions $\boldsymbol{R}_1,\boldsymbol{R}_2,\boldsymbol{\beta}_1,\boldsymbol{\beta}_2,\boldsymbol{Y}_1,\boldsymbol{Y}_2$. That way, Eq. (3.14) can be expressed entirely in terms of the KLDS parameters $\boldsymbol{A},\boldsymbol{B},\boldsymbol{Y},\boldsymbol{R},\boldsymbol{\beta}$. Eq. (3.14) will be the preferred way to write the Alignment distance on pairs of KLDS's in the following.

Metric Property

The Alignment distance, $d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}$, i.e., the square root of Eq. (3.14) is a metric on $\mathcal{RK}_d^n$, if the weights $\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}} > 0$ are all strictly positive. To verify this, its positive definiteness, its symmetry, as well as the triangle equality need to be shown, which is done in the following.

- The Alignment distance is *positive definite*, if all weights are strictly positive, i.e.,

  $$d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1,\Xi_2) \geq 0, \tag{3.17}$$

  with equality being attained, iff there is a matrix $\boldsymbol{Q}\in\mathrm{O}(n)$ such that

  $$\Xi_1 = \boldsymbol{Q}\bullet\Xi_2 \tag{3.18}$$

  holds. Eq. (3.17) is obvious, as all terms in Eq. (3.10) are squared norms. It is also easy to see that the Alignment distance vanishes, if the two systems Ξ_1 and Ξ_2 belong to the same equivalence class with respect to

Eq. (3.7). The opposite implication is also true: If there is no $\boldsymbol{Q} \in \mathrm{O}(n)$, such that each parameter of Ξ_1 and $\boldsymbol{Q} \bullet \Xi_2$ is equal, then at least one of the norms in the weighted sum of Eq. (3.10) is strictly positive, and so is $d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2)$.

- *Symmetry*, i.e., the property

$$d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) = d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_2, \Xi_1) \tag{3.19}$$

is easily shown by observing the form in Eq. (3.14). The first term consisting of the function $\tau_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2)$ as defined in Eq. (3.15) is obviously symmetric. Furthermore, we can derive the equality

$$\begin{aligned}\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q} \bullet \Xi_2) =& \lambda_{\boldsymbol{A}} \operatorname{tr}(\boldsymbol{A}_1^\top \boldsymbol{Q} \boldsymbol{A}_2 \boldsymbol{Q}^\top) + \lambda_{\boldsymbol{B}} \operatorname{tr}(\boldsymbol{B}_1^\top \boldsymbol{Q} \boldsymbol{B}_2) \\ &+ \operatorname{tr}(\boldsymbol{R}_1^\top \kappa(\boldsymbol{Y}_1, \boldsymbol{Y}_2) \boldsymbol{R}_2 \boldsymbol{Q}^\top) \\ =& \lambda_{\boldsymbol{A}} \operatorname{tr}(\boldsymbol{A}_2^\top \boldsymbol{Q}^\top \boldsymbol{A}_1 \boldsymbol{Q}) + \lambda_{\boldsymbol{B}} \operatorname{tr}(\boldsymbol{B}_2^\top \boldsymbol{Q}^\top \boldsymbol{B}_1) \\ &+ \operatorname{tr}(\boldsymbol{R}_2^\top \kappa(\boldsymbol{Y}_2, \boldsymbol{Y}_1) \boldsymbol{R}_1 \boldsymbol{Q}) \\ =& \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_2, \boldsymbol{Q}^\top \bullet \Xi_1)\end{aligned} \tag{3.20}$$

by using invariance properties of the trace function. If $\boldsymbol{Q}$ is a matrix in $\mathrm{O}(n)$, so is $\boldsymbol{Q}^\top$ and thus the equality

$$\max_{\boldsymbol{Q} \in \mathrm{O}(n)} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_2, \boldsymbol{Q}^\top \bullet \Xi_1) = \max_{\boldsymbol{Q} \in \mathrm{O}(n)} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_2, \boldsymbol{Q} \bullet \Xi_1) \tag{3.21}$$

holds, which implies the equation

$$\max_{\boldsymbol{Q} \in \mathrm{O}(n)} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q} \bullet \Xi_2) = \max_{\boldsymbol{Q} \in \mathrm{O}(n)} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_2, \boldsymbol{Q} \bullet \Xi_1). \tag{3.22}$$

Hence, the second term in Eq. (3.14), is also symmetric.

- The triangle inequality can be directly inferred from the fact that

$$d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) = \sqrt{\tau_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) - 2\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \Xi_2)} \tag{3.23}$$

is a metric on $\mathcal{K}_d^n$ and as such abides the triangle inequality. Let Ξ_1, Ξ_2 and Ξ_3 be three systems and $\boldsymbol{Q}_{1,2}, \boldsymbol{Q}_{2,3}$ be the two orthogonal matrices that minimize Eq. (3.10) for the pairs Ξ_1, Ξ_2 and Ξ_2, Ξ_3, respectively. Then, we can deduct

$$\begin{aligned}&d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \Xi_2) + d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_2, \Xi_3) \\ =& d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_1, \boldsymbol{Q}_{1,2} \bullet \Xi_2) + d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}(\Xi_2, \boldsymbol{Q}_{2,3} \bullet \Xi_3).\end{aligned} \tag{3.24}$$

From the properties of the trace function, it should become apparent that

$$\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q} \bullet \Xi_2) = \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\boldsymbol{Q}^\top \bullet \Xi_1, \Xi_2) \tag{3.25}$$

holds for any $\boldsymbol{Q} \in \mathrm{O}(n)$. Thus, we can rewrite Eq. (3.24) as

$$\begin{aligned}&d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\Xi_2)+d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_2,\Xi_3)\\ =&d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\boldsymbol{Q}_{1,2}^{\top}\bullet\Xi_1,\Xi_2)+d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_2,\boldsymbol{Q}_{2,3}\bullet\Xi_3).\end{aligned} \tag{3.26}$$

Due to the triangle inequality of $d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}$, we can also conclude

$$\begin{aligned}&d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\Xi_2)+d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_2,\Xi_3)\\ \geq&d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\boldsymbol{Q}_{1,2}^{\top}\bullet\Xi_1,\boldsymbol{Q}_{2,3}\bullet\Xi_3)\\ =&d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\boldsymbol{Q}_{1,2}\boldsymbol{Q}_{2,3}\bullet\Xi_3).\end{aligned} \tag{3.27}$$

Since $\boldsymbol{Q}_{1,2}\boldsymbol{Q}_{2,3}$ is an orthogonal matrix, $d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\Xi_3)$ is a lower bound for $d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\boldsymbol{Q}_{1,2}\boldsymbol{Q}_{2,3}\bullet\Xi_3)$ by definition, which implies

$$\begin{aligned}&d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\Xi_2)+d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_2,\Xi_3)\\ \geq&d_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi_1,\Xi_3).\end{aligned} \tag{3.28}$$

The LDS version of the Alignment distance in Eq. (2.58) is also a metric on $\mathcal{QO}_d^n$, since the scalar product in $\mathbb{R}^d$ is also a kernel.

Computation

Although it is possible to calculate the Alignment distances of two LDS's by a simple gradient descent method on $\mathrm{O}(n)$ as demonstrated in [2], the authors of [61] observe empirically that a coordinate-descent algorithm yields better convergence properties. The algorithm they propose is adapted for the case of KLDS's and described in the following. Additionally, one simple modification is made: In the version described in [61], the algorithm divides the optimization problem over $\mathrm{O}(n)$ into subproblems by considering each two-dimensional sub-plane of $\mathbb{R}^n$ individually. The sub-planes are chosen following an ascending order of the indexes. In the version presented here, the coordinates are randomly permuted for each iteration. That way, it is easier to show that the algorithm converges to a set of critical points.

An important concept the algorithm builds upon is the *Givens rotation*. The term refers to a matrix $\boldsymbol{G}_{k,l}(c,s)$ that performs a rotation in the plane spanned by the kth and lth coordinates. The parameters $c=\cos(\alpha), s=\sin(\alpha)$ describe the cosine and sine of a rotation angle α, respectively. It has the form

$$\boldsymbol{G}_{k,l}(c,s) = \begin{bmatrix} \ddots & & & & & & & \\ & 1 & & & & & & \\ & & c & & & s & & \\ & & & 1 & & & & \\ & & & & \ddots & & & \\ & & & & & 1 & & \\ & & -s & & & c & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \end{bmatrix}. \tag{3.29}$$

The set $\mathrm{O}(n)$ of $n \times n$-dimensional, orthogonal matrices contains two connected components, the *special orthogonal* group of rotations in $\mathbb{R}^n$ containing orthogonal matrices with a positive determinant, denoted by $\mathrm{SO}(n)$, and the remainder $\mathrm{O}(n) \setminus \mathrm{SO}(n)$ containing orthogonal matrices with a negative determinant. In the following, we exploit the fact that any rotation matrix $\boldsymbol{Q} \in \mathrm{SO}(n)$ can be written as a product of Givens rotations. In fact, the aforementioned sets can be written as

$$\mathrm{SO}(n) = \left\{ \prod_{\substack{k \in \{1,\dots,n-1\}, \\ l \in \{k+1,\dots,n\}}} \boldsymbol{G}_{k,l}(c_{k,l}, s_{k,l}) \,\middle|\, c_{k,l}^2 + s_{k,l}^2 = 1 \;\forall\, k,l \right\}, \tag{3.30}$$

and

$$\mathrm{O}(n) \setminus \mathrm{SO}(n) = \{\boldsymbol{Q}\, \mathrm{diag}\,(-1,1,\dots,1) \mid \boldsymbol{Q} \in \mathrm{SO}(n)\}, \tag{3.31}$$

respectively. This allows us to optimize over $\mathrm{SO}(n)$ and $\mathrm{O}(n) \setminus \mathrm{SO}(n)$ by factorizing rotation matrices into their Givens factors, and then optimizing for each factor individually. Specifically, solving the optimization problem of Eq. (3.14),

$$\max_{\boldsymbol{Q} \in O(n)} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}} (\Xi_1, \boldsymbol{Q} \bullet \Xi_2), \tag{3.32}$$

is more difficult than doing so after restricting the feasible set to Givens rotations for one particular coordinate pair k, l, i.e.,

$$\max_{\substack{c,s \\ \text{s.t. } c^2+s^2}} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}} (\Xi_1, \boldsymbol{G}_{k,l}(c,s) \bullet \Xi_2), \tag{3.33}$$

because Eq. (3.33) can be solved in closed form, as outlined in the following. Using some coefficients $k_0, \dots, k_5 \in \mathbb{R}$, the function in Eq. (3.33) can be rewritten as a bi-quadratic function in c and s, i.e.,

$$\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}} (\Xi_1, \boldsymbol{G}_{k,l}(c,s) \bullet \Xi_2) = k_0 c^2 + k_1 cs + k_2 s^2 + k_3 c + k_4 s + k_5. \tag{3.34}$$

See Appendix A.1 for a derivation. Since s and c are constrained to be the $\sin$ and $\cos$ of an angle, the equality

$$s^2 + c^2 = 1 \tag{3.35}$$

holds. Therefore we can reformulate Eq. (3.34) as a function of $c \in [-1, 1]$, i.e.,

$$h(c) = k_0 c^2 \pm k_1 c \sqrt{1 - c^2} + k_2 (1 - c^2) + k_3 c \pm k_4 \sqrt{1 - c^2} + k_5. \tag{3.36}$$

The expression in Eq. (3.36) can be optimized for c by computing its derivative and locating the zeros of the result. This can be done by solving the *quartic* equation

$$\begin{aligned} -4((k_0 - k_2)^2 + k_1^2)c^4 - 4(k_3(k_0 - k_2) + k_1 k_4)c^3 \\ +(4(k_0 - k_2)^2 + 4k_1^2 - k_3^2 - k_4^2)c^2 + 2(2k_3(k_0 - k_2) + k_1 k_4)c \\ +(k_3^2 - k_1^2) = 0. \end{aligned} \tag{3.37}$$

Appendix A.2 provides a derivation of Eq. (3.37).

The quartic polynomial in Eq. (3.37) can have up to four roots c_1, c_2, c_3, c_4 in the interval of interest $[-1, 1]$ that can be determined using closed-form expressions. This yields up to eight solution candidates for Eq. (3.33), namely

$$\begin{aligned} &\hat{c} = c_1, \hat{s} = \sqrt{1 - c_1^2}, \qquad \hat{c} = c_1, \hat{s} = -\sqrt{1 - c_1^2}, \\ &\hat{c} = c_2, \hat{s} = \sqrt{1 - c_2^2}, \qquad \hat{c} = c_2, \hat{s} = -\sqrt{1 - c_2^2}, \\ &\hat{c} = c_3, \hat{s} = \sqrt{1 - c_3^2}, \qquad \hat{c} = c_3, \hat{s} = -\sqrt{1 - c_3^2}, \\ &\hat{c} = c_4, \hat{s} = \sqrt{1 - c_4^2}, \qquad \hat{c} = c_4, \hat{s} = -\sqrt{1 - c_4^2}. \end{aligned} \tag{3.38}$$

Additionally, there is also the possibility that the maximum of $h(c)$ in $[-1, 1]$ is on one of the two boundaries of the feasible interval. It is thus also possible that it is maximized by

$$\hat{c} = -1, \qquad \hat{s} = 0 \tag{3.39}$$

or

$$\hat{c} = 1, \qquad \hat{s} = 0. \tag{3.40}$$

In order to solve the optimization problem Eq. (3.33), the feasible candidate pair $\hat{c}, \hat{s}$ that maximizes $h(c)$ is chosen.

Being able to optimize Eq. (3.34) enables us to formulate the procedure in Algorithm 7 to solve the overall problem Eq. (3.32). In each iteration, the algorithm sweeps through all possible coordinate pairs k, l to find a Givens factorization that maximizes the objective function coordinate wise.

Algorithm 7 : Coordinate descent algorithm for solving Eq. (3.32)

```
   Input : Systems
           $\Xi_1 = (\boldsymbol{A}_1, \boldsymbol{B}_1, \boldsymbol{R}_1, \boldsymbol{\beta}_1, \boldsymbol{Y}_1) \in \mathcal{K}_d^n$, $\Xi_2 = (\boldsymbol{A}_2, \boldsymbol{B}_2, \boldsymbol{R}_2, \boldsymbol{\beta}_2, \boldsymbol{Y}_2) \in \mathcal{K}_d^n$
1  Initialize $\boldsymbol{Q}_+ \in \mathrm{SO}(n)$;
2  while $\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}\left(\Xi_1, \tilde{\Xi}_2\right)$ not converged do
3      for random $k \in \{1, \dots, n-1\}$ do
4          for random $l \in \{k, \dots, n\}$ do
5              $\tilde{\Xi}_2 \leftarrow \boldsymbol{Q}_+ \bullet \Xi_2$;
6              Compute $k_0, \dots, k_4$ in Eq. (3.34) for $\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q}_+ \bullet \tilde{\Xi}_2)$;
7              Generate $c, s$ candidates via Eq. (3.37)-Eq. (3.40);
8              $\hat{c}, \hat{s} \leftarrow \arg\max_{(c,s)} k_0 c^2 + k_1 cs + k_2 s^2 + k_3 c + k_4 s + k_5$;
9              $\boldsymbol{Q}_+ \leftarrow \boldsymbol{G}_{k,l}(\hat{c}, \hat{s}) \boldsymbol{Q}_+$;
10         end
11     end
12 end
13 Initialize $\boldsymbol{Q}_- \in \mathrm{O}(n) \setminus \mathrm{SO}(n)$;
14 Repeat while loop for $\boldsymbol{Q}_-$;
15 $\hat{\boldsymbol{Q}} \leftarrow \arg\max_{\boldsymbol{Q} \in \{\boldsymbol{Q}_+, \boldsymbol{Q}_-\}} \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q} \bullet \Xi_2)$;
   Output : $\hat{\boldsymbol{Q}}$
```

Convergence

As the problem in Eq. (3.14) is non-convex, it is difficult to provide global convergence guarantees. However, it is possible to show that any solution $\hat{\boldsymbol{Q}}$ returned by Algorithm 7 will almost certainly be (close to a) critical point of the objective function. This is stated in the following theorem.

Theorem 1. *Let $(\boldsymbol{Q}^{(i)})_{i \in \mathbb{N}}$ be the sequence of orthogonal matrices generated by the **while** loop in Algorithm 7. It is bounded with respect to the Frobenius norm and thus has at least one accumulation point. Furthermore, every accumulation point of the sequence is almost certainly a critical point of the smooth objective function*

$$L_{\mathrm{align}}(\boldsymbol{Q}) = \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q} \bullet \Xi_2). \tag{3.41}$$

Proof. See Appendix A.3. □

Theorem 1 essentially states that, no matter the initialization of Algorithm 7, $(\boldsymbol{Q}^{(i)})_{i \in \mathbb{N}}$ converges (almost certainly) to a set of critical points. In other words, if we denote the set containing the critical points of L_{align} by $\mathcal{C}$, the sequence

$\min_{\boldsymbol{Q}\in\mathcal{C}} \|\boldsymbol{Q}^{(i)} - \boldsymbol{Q}\|_F$ will tend towards 0. If this was not the case, this sequence would have a subsequence with all elements being larger than a certain lower bound $\epsilon > 0$, which is not possible since every subsequence of $\boldsymbol{Q}^{(i)}$ has an accumulation point that is (almost certainly) a critical point.

Theoretically, this critical point could be a saddle point, but it is much more likely to be a (local) maximum of the objective function, as saddle points tend to be unstable.

3.3. Averages of KLDS Sets

Fréchet Means

Certain problems, such as clustering or classification of data points rely on the computation of arithmetic means from data sets. In the case of KLDS's, this is generally not possible since the set $\mathcal{K}_d^n$ is not a vector space. An alternative approach is to find an appropriate generalization of arithmetic means to finite subsets of $\mathcal{K}_d^n$. *Fréchet means* generalize arithmetic means to any set $\mathcal{S}$ equipped with a distance function $d_{\mathcal{S}}(\cdot,\cdot)$. The Fréchet mean $\bar{\boldsymbol{s}}$ of a finite subset $\{\boldsymbol{s}_1,\dots,\boldsymbol{s}_K\}$ of $\mathcal{S}$ is defined as the minimizer of the sum of squared distances to $\boldsymbol{s}_1,\dots,\boldsymbol{s}_K$, i.e.,

$$\bar{\boldsymbol{s}} = \arg\min_{\boldsymbol{s}\in\mathcal{S}} \sum_{i=1}^{K} d_{\mathcal{S}}^2(\boldsymbol{s},\boldsymbol{s}_i). \tag{3.42}$$

The Fréchet mean is not only a sensible definition for sets that do not have the additivity structure of vector spaces, but also a suitable choice for representative points in classification scenarios. This is in particular true, if d is a metric on $\mathcal{S}$. To see why this is the case, consider the situation where we have a test point $\boldsymbol{s}_{\text{test}}$, and we want to measure its proximity to the set containing $\boldsymbol{s}_1,\dots,\boldsymbol{s}_K$. The aim is to assess if it belongs to the same semantic class as $\boldsymbol{s}_1,\dots,\boldsymbol{s}_K$. This could be done by 1-NN, where the distance to the set is determined by the point $\boldsymbol{s}_i, i\in\{1,\dots,K\}$ closest to $\boldsymbol{s}_{\text{test}}$. If we want to avoid storing the entire training data set in memory, we could refrain to NCC classification by choosing a representative point $\boldsymbol{s}_{\text{rep}}$ for each class. This point is then stored instead of all the samples contained in the respective class. If $d_{\mathcal{S}}$ is a metric, then the triangle inequality yields

$$\begin{aligned} d_{\mathcal{S}}(\boldsymbol{s}_{\text{test}},\boldsymbol{s}_{\text{rep}}) - d_{\mathcal{S}}(\boldsymbol{s}_{\text{rep}},\boldsymbol{s}_i) &\le d_{\mathcal{S}}(\boldsymbol{s}_{\text{test}},\boldsymbol{s}_i) \\ &\le d_{\mathcal{S}}(\boldsymbol{s}_{\text{test}},\boldsymbol{s}_{\text{rep}}) + d_{\mathcal{S}}(\boldsymbol{s}_{\text{rep}},\boldsymbol{s}_i), \end{aligned} \tag{3.43}$$

such that a small value for $d_{\mathcal{S}}(\boldsymbol{s}_{\text{rep}},\boldsymbol{s}_i)$ for any possible i implies that $d_{\mathcal{S}}(\boldsymbol{s}_{\text{test}},\boldsymbol{s}_{\text{rep}})$ is always a good approximation for $d_{\mathcal{S}}(\boldsymbol{s}_{\text{test}},\boldsymbol{s}_i)$. This motivates choosing $\boldsymbol{s}_{\text{rep}} = \bar{\boldsymbol{s}}$ according to Eq. (3.42), because it minimizes $d_{\mathcal{S}}(\boldsymbol{s}_{\text{rep}},\boldsymbol{s}_i)$ for all i in the sense of the mean squared error. The motivation is visualized in Figure 3.2.

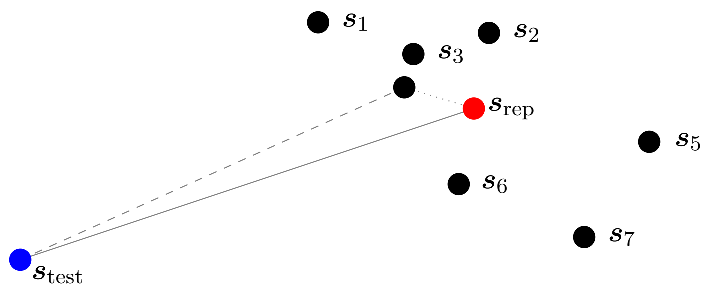


Figure 3.2.: Motivation for using Fréchet means: For a set of reference points (black), the representative point (red) should be chosen such that it its distance to each one of them (dotted) is small. Then, the distance of a test point (blue) to the representative point (continuous) differs little from its distance to any of the reference points represented by the representative point (dashed).

Computation

In [2], an alternating algorithm is presented that approximates the Fréchet means of LDS sets with respect to the Alignment distance by averaging each parameter in every iteration. We cannot directly apply the procedure to KLDS's without further adaptation, since the Fréchet means for the $\boldsymbol{Y}$, $\boldsymbol{R}$ and $\boldsymbol{\beta}$ parameters cannot be computed by simple arithmetic averaging. The following discussion is thus not just a revision of the original algorithm in [2], but a generalization of the procedure to KLDS's. Remarkably, this generalization takes care of the scalability problems caused by the fact that kernel-based machine learning techniques often require the entire training data set for computing representations in the kernel feature space.

Consider a finite set $\{\Xi_i\}_{i\in\{1,\ldots,K\}}$ of which the Fréchet mean $\bar{\Xi} = (\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{R}}, \bar{\boldsymbol{\beta}}, \bar{\boldsymbol{Y}})$ can be written as

$$\bar{\Xi} = \underset{\Xi\in\mathcal{K}_d^n}{\arg\min}\, g(\Xi) \tag{3.44}$$

with

$$\begin{aligned} g(\Xi) &= \sum_{i=1}^{K} d^2_{\text{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi, \Xi_i) \\ &= \sum_{i=1}^{K} d^2_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi, \boldsymbol{Q}_i \bullet \Xi_i), \end{aligned} \tag{3.45}$$

where $\boldsymbol{Q}_i$ is the orthogonal matrix that minimizes $d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\boldsymbol{\varphi}}}}(\Xi, \boldsymbol{Q} \bullet \Xi_i)$ for each $i \in \{1, \ldots, K\}$.

The alternating algorithm works as follows. In each iteration, the Fréchet mean of $\{\boldsymbol{Q}_i \bullet \Xi_i\}_{i\in\{1,\ldots,K\}}$ with respect to $d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}$ is determined, and the results are used to compute the Alignment distances of the current estimation of $\bar{\Xi}$ to the systems $\Xi_1,\ldots,\Xi_K$, yielding new estimations for the orthogonal matrices $\boldsymbol{Q}_1,\ldots,\boldsymbol{Q}_K$. These two steps are repeated, until $g(\Xi)$ is no longer significantly decreased.

Computing the Fréchet mean with regards to $d_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}$ can be performed for the parameters $\boldsymbol{A}$ and $\boldsymbol{B}$, separately, as described in [2]. For the latter, it can be carried out by simply computing

$$\begin{aligned}\bar{\boldsymbol{B}} &= \arg\min_{\boldsymbol{B}} g((\boldsymbol{A},\boldsymbol{B},\boldsymbol{R},\boldsymbol{\beta},\boldsymbol{Y}))\\ &= \arg\min_{\boldsymbol{B}} \sum_{i=1}^{K} d^2_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}((\boldsymbol{A},\boldsymbol{B},\boldsymbol{R},\boldsymbol{\beta},\boldsymbol{Y}),\boldsymbol{Q}_i \bullet \Xi_i)\\ &= \arg\min_{\boldsymbol{B}} \sum_{i=1}^{K} \|\boldsymbol{B}-\boldsymbol{Q}_i\boldsymbol{B}_i\|_F^2\\ &= \frac{1}{K}\sum_{i=1}^{N} \boldsymbol{Q}_i\boldsymbol{B}_i.\end{aligned} \tag{3.46}$$

Likewise, the arithmetic average (and thus the Fréchet mean) for $\boldsymbol{A}$ is given by

$$\begin{aligned}\bar{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} g((\boldsymbol{A},\boldsymbol{B},\boldsymbol{R},\boldsymbol{\beta},\boldsymbol{Y}))\\ &= \arg\min_{\boldsymbol{A}} \sum_{i=1}^{K} d^2_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}((\boldsymbol{A},\boldsymbol{B},\boldsymbol{R},\boldsymbol{\beta},\boldsymbol{Y}),\boldsymbol{Q}_i \bullet \Xi_i)\\ &= \arg\min_{\boldsymbol{A}} \sum_{i=1}^{K} \|\boldsymbol{A}-\boldsymbol{Q}_i\boldsymbol{A}_i\boldsymbol{Q}_i^\top\|_F^2\\ &= \frac{1}{K}\sum_{i=1}^{N} \boldsymbol{Q}_i\boldsymbol{A}_i\boldsymbol{Q}_i^\top.\end{aligned} \tag{3.47}$$

By definition, any state transition matrix of a KLDS in $\mathcal{K}_n^d$ has a spectral radius smaller than 1. Due to the triangle inequality and the fact that orthogonal similarity (congruence) transformations do not affect the spectrum of a matrix, the upper bound

$$\|\bar{\boldsymbol{A}}\|_2 \leq \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{Q}_i\boldsymbol{A}_i\boldsymbol{Q}_i^\top\|_2 = \frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{A}_i\|_2 < 1 \tag{3.48}$$

holds. The stability of the resulting system is thus not affected.

The remaining parameters need to fulfill the condition

$$\bar{\boldsymbol{\beta}}, \bar{\boldsymbol{R}}, \bar{\boldsymbol{Y}} = \underset{\substack{\boldsymbol{\beta}\in\mathbb{R}^{\bar{N}}, \boldsymbol{R}\in\mathbb{R}^{\bar{N}\times n},\\ \boldsymbol{Y}\in\mathbb{R}^{d\times\bar{N}}}}{\arg\min} g((\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{R}, \boldsymbol{\beta}, \boldsymbol{Y})). \tag{3.49}$$

Equivalently, we can write

$$\begin{aligned}\bar{\boldsymbol{\beta}}, \bar{\boldsymbol{R}}, \bar{\boldsymbol{Y}} = &\underset{\substack{\boldsymbol{\beta}\in\mathbb{R}^{\bar{N}}, \boldsymbol{R}\in\mathbb{R}^{\bar{N}\times n},\\ \boldsymbol{Y}\in\mathbb{R}^{d\times\bar{N}}}}{\arg\min} \sum_{i=1}^{K} d^2_{F,\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}((\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{R}, \boldsymbol{\beta}, \boldsymbol{Y}), \boldsymbol{Q}_i \bullet \Xi_i)\\ = &\underset{\substack{\boldsymbol{\beta}\in\mathbb{R}^{\bar{N}}, \boldsymbol{R}\in\mathbb{R}^{\bar{N}\times n},\\ \boldsymbol{Y}\in\mathbb{R}^{d\times\bar{N}}}}{\arg\max} 2\sum_{i=1}^{K} \operatorname{tr}(\boldsymbol{R}^\top \kappa(\boldsymbol{Y}, \boldsymbol{Y}_i)\boldsymbol{R}_i\boldsymbol{Q}_i^\top)\\ &\quad + \lambda_{\bar{\varphi}}\left(2\sum_{i=1}^{K} \boldsymbol{\beta}^\top \kappa(\boldsymbol{Y}, \boldsymbol{Y}_i)\boldsymbol{\beta}_i - K\boldsymbol{\beta}^\top\kappa(\boldsymbol{Y}, \boldsymbol{Y})\boldsymbol{\beta}\right)\\ &\text{s.t. } \bar{N} \in \mathbb{N},\ \boldsymbol{R}^\top\kappa(\boldsymbol{Y}, \boldsymbol{Y})\boldsymbol{R} = \boldsymbol{I}_n.\end{aligned} \tag{3.50}$$

Let us assume that we have determined the solution for $\bar{\boldsymbol{Y}} \in \mathbb{R}^{\bar{N}}$. In that case, we can provide closed-form solutions for the remaining parameters, as explained in the following. Let the sequence lengths of the systems $\Xi_1, \ldots, \Xi_K$ be denoted by $N_1, \ldots, N_K$, respectively. First, note that there is a positive integer N_{all} with

$$\bar{N} \leq N_{\text{all}} \leq N_1 + \cdots + N_K + \bar{N}, \tag{3.51}$$

such that there must be a matrix $\boldsymbol{G} \in \mathbb{R}^{N_{\text{all}}\times\bar{N}}$ and matrices $\boldsymbol{H}_1 \in \mathbb{R}^{N_{\text{all}}\times N_1}, \ldots, \boldsymbol{H}_K \in \mathbb{R}^{N_{\text{all}}\times N_K}$ for which the equations

$$\begin{aligned}\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}}) &= \boldsymbol{G}^\top\boldsymbol{G},\\ \kappa(\bar{\boldsymbol{Y}}, \boldsymbol{Y}_i) &= \boldsymbol{G}^\top\boldsymbol{H}_i,\\ \kappa(\boldsymbol{Y}_i, \boldsymbol{Y}_j) &= \boldsymbol{H}_i^\top\boldsymbol{H}_j\end{aligned} \tag{3.52}$$

are satisfied, for all $i, j \in \{1, \ldots, K\}$. This is due to the existence of an N_{all}-dimensional subspace of $\mathbb{H}$ that contains the feature representations of all the columns in $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_K$ and $\bar{\boldsymbol{Y}}$.

Therefore, using Eq. (3.50), we write

$$\begin{aligned}\bar{\boldsymbol{\beta}} &= \underset{\boldsymbol{\beta}\in\mathbb{R}^{\bar{N}}}{\arg\max}\, 2\boldsymbol{\beta}^\top \sum_{i=1}^{K} \kappa(\bar{\boldsymbol{Y}}, \boldsymbol{Y}_i)\boldsymbol{\beta}_i - K\boldsymbol{\beta}^\top\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}})\boldsymbol{\beta}\\ &= \underset{\boldsymbol{\beta}\in\mathbb{R}^{\bar{N}}}{\arg\max}\, 2\boldsymbol{\beta}^\top\boldsymbol{G}^\top \sum_{i=1}^{K} \boldsymbol{H}_i\boldsymbol{\beta}_i - K\boldsymbol{\beta}^\top\boldsymbol{G}^\top\boldsymbol{G}\boldsymbol{\beta}.\end{aligned} \tag{3.53}$$

Setting the derivative of the objective in Eq. (3.53) to 0 yields

$$\bar{\boldsymbol{\beta}} = \frac{1}{K}\boldsymbol{G}^{+}\sum_{i=1}^{K}\boldsymbol{H}_i\boldsymbol{\beta}_i, \tag{3.54}$$

which can be also written as

$$\begin{aligned}\bar{\boldsymbol{\beta}} &= \frac{1}{K}(\boldsymbol{G}^\top\boldsymbol{G})^{+}\boldsymbol{G}^\top\sum_{i=1}^{K}\boldsymbol{H}_i\boldsymbol{\beta}_i \\ &= \frac{1}{K}\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}})^{+}\sum_{i=1}^{K}\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_i)\boldsymbol{\beta}_i.\end{aligned} \tag{3.55}$$

Similarly, for $\bar{\boldsymbol{R}}$, we have

$$\bar{\boldsymbol{R}} = \underset{\substack{\boldsymbol{R}\in\mathbb{R}^{\bar{N}\times n} \\ \text{s.t. } \boldsymbol{R}^\top\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}})\boldsymbol{R}=\boldsymbol{I}_n}}{\arg\max} \sum_{i=1}^{K}\operatorname{tr}(\boldsymbol{R}^\top\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_i)\boldsymbol{R}_i\boldsymbol{Q}_i^\top). \tag{3.56}$$

Let us denote the SVD of $\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}})$ by

$$\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}}) = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top, \tag{3.57}$$

and its rank by $r \leq \bar{N}$. Due to the orthogonality requirement in Eq. (3.5), the matrix $\bar{\boldsymbol{R}}$ must have the form

$$\bar{\boldsymbol{R}} = \boldsymbol{U}_{(r)}\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}}\bar{\boldsymbol{W}}, \tag{3.58}$$

where $\bar{\boldsymbol{W}} \in \operatorname{St}(n,r)$ is a matrix with orthonormal columns, such that

$$\bar{\boldsymbol{R}}^\top\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}})\bar{\boldsymbol{R}} = \boldsymbol{I}_n \tag{3.59}$$

always holds. This allows us to rewrite Eq. (3.56) yielding the alternative formulation

$$\bar{\boldsymbol{W}} = \underset{\boldsymbol{W}\in\operatorname{St}(n,r)}{\arg\max}\ \operatorname{tr}\left(\boldsymbol{W}^\top\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}}\boldsymbol{U}_{(r)}^\top\sum_{i=1}^{K}\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_i)\boldsymbol{R}_i\boldsymbol{Q}_i^\top\right), \tag{3.60}$$

which is solved by choosing $\boldsymbol{W}$ such that the argument of the trace function is symmetric. This is obtained by computing the SVD

$$\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}}\boldsymbol{U}_{(r)}^\top\sum_{i=1}^{K}\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_i)\boldsymbol{R}_i\boldsymbol{Q}_i^\top = \boldsymbol{U}'\boldsymbol{\Sigma}'\boldsymbol{V}'^\top, \tag{3.61}$$

and setting

$$\bar{\boldsymbol{W}} = \boldsymbol{U}'_{(n)}\boldsymbol{V}'^\top. \tag{3.62}$$

If we now substitute these results into the objective Eq. (3.50), we get

$$\begin{aligned}
&2\sum_{i=1}^{K}\operatorname{tr}(\bar{\boldsymbol{R}}^\top\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_i)\boldsymbol{R}_i\boldsymbol{Q}_i^\top)\\
&+\lambda_{\bar{\varphi}}\left(2\sum_{i=1}^{K}\bar{\boldsymbol{\beta}}^\top\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_i)\bar{\boldsymbol{\beta}}_i-K\bar{\boldsymbol{\beta}}^\top\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}})\boldsymbol{\beta}\right)\\
=&2\sum_{i=1}^{K}\operatorname{tr}(\boldsymbol{V}'\boldsymbol{\Sigma}'^{(n)}_{(n)}\boldsymbol{V}'^\top)+\frac{\lambda_{\bar{\varphi}}}{K}\sum_{\substack{i=1,\\ j=1}}^{K}\boldsymbol{\beta}_i^\top\kappa(\boldsymbol{Y}_i,\bar{\boldsymbol{Y}})\kappa(\bar{\boldsymbol{Y}},\bar{\boldsymbol{Y}})^+\kappa(\bar{\boldsymbol{Y}},\boldsymbol{Y}_j)\boldsymbol{\beta}_j.
\end{aligned}\tag{3.63}$$

Now that we have obtained the solutions $\bar{\boldsymbol{\beta}}$ and $\bar{\boldsymbol{R}}$, let us consider the choice of $\bar{\boldsymbol{Y}}$. Let us write

$$\boldsymbol{U}_G=\boldsymbol{G}\boldsymbol{U}_{(r)}\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}}\in\operatorname{St}(r,N_{\text{all}}).\tag{3.64}$$

Since $\boldsymbol{V}'$ is orthogonal, the term $\operatorname{tr}(\boldsymbol{V}'\boldsymbol{\Sigma}'^{(n)}_{(n)}\boldsymbol{V}'^\top)$ is equal to the sum of the singular values of Eq. (3.61) in $\boldsymbol{\Sigma}'^{(n)}_{(n)}$. Rewriting the trace as a nuclear norm $\|\cdot\|_*$ and the kernel matrices by means of $\boldsymbol{G}$ and $\boldsymbol{H}_i$ yields the equivalent formulation

$$\begin{aligned}
&2\left\|\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}}\boldsymbol{U}_{(r)}^\top\sum_{i=1}^{K}\boldsymbol{G}^\top\boldsymbol{H}_i\boldsymbol{R}_i\boldsymbol{Q}_i^\top\right\|_*+\frac{\lambda_{\bar{\varphi}}}{K}\sum_{\substack{i=1,\\ j=1}}^{K}\boldsymbol{\beta}_i^\top\boldsymbol{H}_i^\top\boldsymbol{G}(\boldsymbol{G}^\top\boldsymbol{G})^+\boldsymbol{G}^\top\boldsymbol{H}_j\boldsymbol{\beta}_j\\
=&2\left\|\boldsymbol{U}_G^\top\sum_{i=1}^{K}\boldsymbol{H}_i\boldsymbol{R}_i\boldsymbol{Q}_i^\top\right\|_*+\frac{\lambda_{\bar{\varphi}}}{K}\sum_{\substack{i=1,\\ j=1}}^{K}\boldsymbol{\beta}_i^\top\boldsymbol{H}_i^\top\boldsymbol{U}_G\boldsymbol{U}_G^\top\boldsymbol{H}_j\boldsymbol{\beta}_j.
\end{aligned}\tag{3.65}$$

If we want to maximize this term for $\boldsymbol{U}_G$, we can do so by choosing it in a way so that the columns of $\boldsymbol{H}_1,\dots,\boldsymbol{H}_N$ are contained in its column space. This can be done by choosing $\bar{\boldsymbol{Y}}$ such that it contains the columns of $\boldsymbol{Y}_1,\dots,\boldsymbol{Y}_K$, i.e.

$$\bar{\boldsymbol{Y}}=\begin{bmatrix}\boldsymbol{Y}_1 & \cdots & \boldsymbol{Y}_K\end{bmatrix}\in\mathbb{R}^{d\times N_1+\dots+N_K}.\tag{3.66}$$

However, this would imply that the Fréchet mean needs as much memory and computational power to be processed as the entirety of sequences represented by it. If we cannot choose the optimal solution in Eq. (3.66) for $\bar{\boldsymbol{Y}}$, the optimality gap can be kept small, by keeping the subspace spanned by the columns of $\boldsymbol{G}$ as close as possible to the column space of $[\boldsymbol{H}_1,\dots,\boldsymbol{H}_K]$. Heuristically, it has been argued that Nyström interpolation [40] has done a good job in achieving this goal. Inspired by the results in [132], $\bar{\boldsymbol{Y}}$ is chosen by employing k-means clustering to the columns of $\boldsymbol{Y}_1,\dots,\boldsymbol{Y}_K$. These considerations altogether yield Algorithm 8. One should keep in mind, that Fréchet means are not necessarily unique.

Algorithm 8 : Fréchet mean computation

```
Input : Set of KLDS descriptors $\{\Xi_i = (\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{R}_i, \boldsymbol{\beta}_i, \boldsymbol{Y}_i)\}_{i\in\{1,\dots,K\}} \subset \mathcal{K}_d^n$,
        sampling size $\bar{N} \in \mathbb{N}$
1  $\bar{\boldsymbol{Y}} \leftarrow k - \mathrm{means}\left(\begin{bmatrix}\boldsymbol{Y}_1 & \cdots & \boldsymbol{Y}_K\end{bmatrix}, \bar{N}\right)$;
2  $\bar{\boldsymbol{\beta}} \leftarrow \frac{1}{K}\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}})^+ \sum_{i=1}^{K} \kappa(\bar{\boldsymbol{Y}}, \boldsymbol{Y}_i)\boldsymbol{\beta}_i$;            // Eq. (3.55)
3  $\boldsymbol{U}, \boldsymbol{\Lambda}, \_ \leftarrow \mathrm{SVD}(\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}}))$;
4  foreach $i \in \{1, \dots, K\}$ do
5      $\boldsymbol{Q}_i \leftarrow \boldsymbol{I}_n$;                         // Initialization
6  end
7  while $g(\bar{\Xi})$ not converged do
8      $\bar{\boldsymbol{A}} \leftarrow \frac{1}{K}\sum_{i=1}^{N} \boldsymbol{Q}_i \boldsymbol{A}_i \boldsymbol{Q}_i^\top$;              // Eq. (3.47)
9      $\bar{\boldsymbol{B}} \leftarrow \frac{1}{K}\sum_{i=1}^{N} \boldsymbol{Q}_i \boldsymbol{B}_i$;                   // Eq. (3.46)
10     $\boldsymbol{U}', \_, \boldsymbol{V}'^\top \leftarrow \mathrm{SVD}\left(\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}} \boldsymbol{U}_{(r)}^\top \sum_{i=1}^{K} \kappa(\bar{\boldsymbol{Y}}, \boldsymbol{Y}_i)\boldsymbol{R}_i \boldsymbol{Q}_i^\top\right)$;
        // Eq. (3.61), $r$ is rank of $\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}})$
11     $\bar{\boldsymbol{R}} \leftarrow \boldsymbol{U}_{(r)} \boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}} \boldsymbol{U}'_{(n)} \boldsymbol{V}'^\top$;              // Eq. (3.62)
12     $\bar{\Xi} \leftarrow (\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{R}}, \bar{\boldsymbol{\beta}}, \bar{\boldsymbol{Y}})$;
13     foreach $i \in \{1, \dots, K\}$ do
14         $\boldsymbol{Q}_i \leftarrow \arg\max_{\boldsymbol{Q}\in \mathrm{O}(n)} \rho_{\lambda_{\boldsymbol{A}}, \lambda_{\boldsymbol{B}}}(\bar{\Xi}, \boldsymbol{Q} \bullet \Xi_i)$;      // Algorithm 7
15     end
16 end
Output : Estimated Fréchet mean $\bar{\Xi}$
```

3.4. Experiments

Overview

Matlab simulations on datasets of dynamic textures and dynamic scenes have been carried out to evaluate the proposed methods for 1-NN and NCC classification. The frame-wise histogram representation was chosen based on its performance in still-image recognition tasks. Dynamic texture frames were encoded by means of LBP [113], while the still images of the dynamic scene videos are encoded via BoW. The χ^2 kernel [44], defined as

$$\kappa_{\chi^2}(\boldsymbol{y}_1, \boldsymbol{y}_2) = \exp\left(-\frac{1}{2} \sum_{\substack{i\in\{1,\dots,d\} \\ \text{s.t. } (\boldsymbol{y}_1)_i + (\boldsymbol{y}_2)_i > 0}} \frac{((\boldsymbol{y}_1)_i - (\boldsymbol{y}_2)_i)^2}{(\boldsymbol{y}_1)_i + (\boldsymbol{y}_2)_i}\right) \tag{3.67}$$

was used for all experiments.

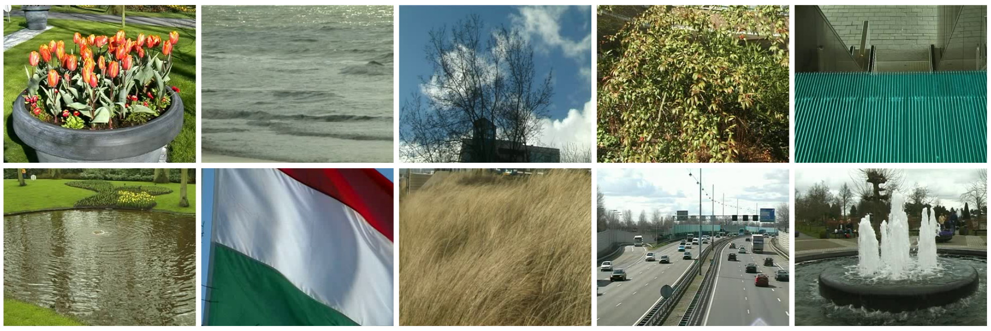


Figure 3.3.: Video frames from the DynTex Gamma split

For the KLDS parameters extracted in this way, the performance of the Alignment distance is compared against the Max SV distance and the Martin distance on 1-NN classification tasks. Apart from this ablation study, 1-NN and NCC results are compared to competing approaches in dynamic texture and dynamic scene classification from computer vision literature. For all experiments, process noise is neglected, i.e., the $\lambda_{\boldsymbol{B}}$ parameter of the Alignment distance was set to 0.

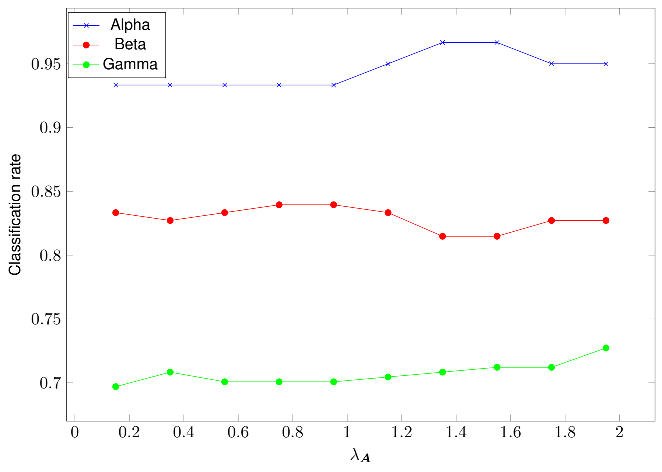
Dynamic Textures

The DynTex database [95] is a collection of high-resolution RGB texture videos. Three splits have been compiled for classification benchmarking.

- *DynTex Alpha* is composed of 60 videos divided into the 3 classes Sea (20 videos), Grass (20), and Trees (20)
- *DynTex Beta* is composed of 162 videos divided into the 10 classes Sea (20), Vegetation (20), Trees (20), Flags (20), Calm Water (20), Fountains (20), Smoke (16), Escalator (7), Traffic (9) and Rotation (10).
- *DynTex Gamma* is composed of 264 videos divided into the 10 classes Flowers (29), Sea (38), Naked trees (25), Foliage (35), Escalator (7), Calm water (30), Flags (31), Grass (23), Traffic (9) and Fountains (37).

Figure 3.3 depicts one video frame from each class of the Gamma split. Some works, like [41] or [134], report their results on a version of *DynTex Gamma* that contains 275 videos, which suggests that there are two versions of this split. In the comparisons, only results that have been reported for the 264-video version are taken into account.

After converting the videos to grayscale, an LBP histogram was computed by means of a third-party tool [113]. From the resulting descriptors, KLDS parameters



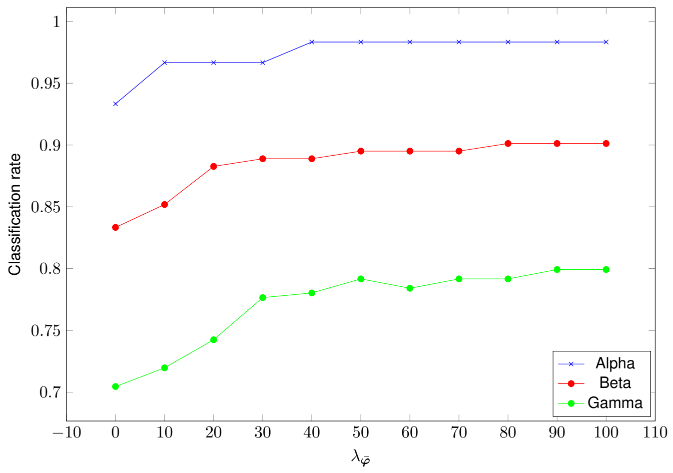


Figure 3.4.: 1-NN classification performance for different weights $\lambda_{\boldsymbol{A}}, \lambda_{\bar{\varphi}}$

for $n = 5$ were computed by means of Algorithm 6. To investigate the impact of the Alignment distance weighting parameters, 1-NN classification has been performed for varying values of $\lambda_{\boldsymbol{A}}$ and $\lambda_{\bar{\varphi}}$. Figure 3.4 depicts the success rate plotted for different values of $\lambda_{\boldsymbol{A}}$ $(\lambda_{\bar{\varphi}} = 0)$ and $\lambda_{\bar{\varphi}}$ $(\lambda_{\boldsymbol{A}} = 0.25)$.

In order to perform NCC classification, Alignment distance Fréchet means have been computed by means of Algorithm 8. As Max SV and Martin distance baselines, medoids were used. For the sake of simplicity, $\lambda_{\bar{\varphi}}$ has been set to 0 for all NCC experiments. The value for $\lambda_{\boldsymbol{A}}$ has been chosen to be 0.25. The parameter λ of the Max SV distance has been determined via grid search for each experiment.

Table 3.1 shows the 1-NN $(\lambda_{\boldsymbol{A}} = 0.25, \lambda_{\bar{\varphi}} = 100.0)$ and NCC $(\lambda_{\boldsymbol{A}} = 0, 25, \lambda_{\bar{\varphi}} = 0)$ classification results in comparison with LBP-TOP [133, 96], Aggregated Salient

	Alpha		Beta		Gamma	
	1-NN	NCC	1-NN	NCC	1-NN	NCC
LBP-TOP	96.7 %	-	85.8 %	-	84.9 %	-
ASF-TOP	91.7 %	-	86.4 %	-	89.4 %	-
MBSIF-TOP	90.0 %	-	90.7 %	-	-*	-
PCANet-Top	96.7 %	-	90.7 %	-	89.4 %	-
DFS	-	83.6 %	-	65.2 %	-	60.8 %
2D+T Curvelet	-	85.0 %	-	67.0 %	-	-*
OTDL	-	86.6 %	-	69.0 %	-	64.2 %
CLSP-TOP	95.0 %	-	92.0 %	-	**91.3** %	-
STRF N-jet	**100.0 %**	-	93.8%	-	91.2 %	-
B3DF	96.7 %	90.0 %	90.1 %	74.1 %	-*	-*
SOE-NET	98.3 %	**96.7 %**	**96.9 %**	**86.4 %**	-*	-*
SoB + Martin	91.7%	83.3 %	74.7 %	51.9 %	64.4 %	41.7 %
SoB + Max SV	96.7 %	85.0 %	84.0 %	59.9 %	78.4 %	54.9 %
SoB + Align	98.3 %	88.3 %	90.1 %	75.3 %	79.9 %	**67.1 %**

*Reported results refer to 275-video version of DynTex Gamma.

Table 3.1.: Recognition rate on DynTex subsets

Features in three orthogonal planes (ASF-TOP) [56], Multiscale Binarized Statistical Image Features in Three Orthogonal Planes (MBSIF-TOP) [10], PCANet-TOP [9], Dynamic Fractal Spectrum (DFS) [128], Spatiotemporal Curvelet Transform (2D+T Curvelet) [41], Orthogonal Tensor Dictionary Learning (OTDL) [98] Completed Local Structure Patterns in Three Orthogonal Planes (CLSP-TOP) [88], Spatio-temporal Receptive Fields (STRF N-jet) [59], Binarized 3D features (B3DF) [134] and Spatiotemporal Oriented Energy Network (SOE-NET) [48]. Only results for grayscale videos are included in the list.

The proposed Alignment distance on KLDS's improves significantly the performance for SoB classification compared to the other two common choices of distance measures, be it for 1-NN or for NCC.

When comparing to other SOTA approaches, the 1-NN classification performance produces competitive results on the simple Alpha split, but falls behind some of the more recent works on the Beta and Gamma split. However, SoB+Align performs comparably well on the NCC experiment, yielding the third-best results in the Alpha split, the second-best results on the Beta split, and the best results on the Gamma split.

Figure 3.5.: Video frames from the YUPENN dataset

Martin	Max SV	Align	SOE	TSVQ	BoST	st-TCoF
83.3 %	80.7 %	86.4 %	74 %	69 %	85 %	**98 %**

Table 3.2.: Recognition rate on YUPENN dataset

Dynamic Scenes

The YUPENN dataset [34] is comprised of the 14 dynamic scene classes Beach, Elevator, Forest fire, Fountain, Highway, Lightning Storm, Ocean, Railway, Rushing river, Clouds, Snowing, City street, Waterfall and Windmill farm. Each class contains 30 RGB videos of different lengths and resolutions, collected from publicly available online resources, such as streaming services.

A BoW histogram of SURF features has been computed for each video frame via OpenCV. The codebooks of size $d = 500$ have been directly computed from the YUPENN videos themselves. To avoid any advantage from learning the codebook on test samples, the dataset was divided into three equally sized parts. For each sample the classification experiment was performed by learning the codebooks from the two parts not including the sample itself.

The resulting streams of histograms were used to compute SoB KLDS's with latent space dimension $n = 10$. The Alignment distance weights were fixed to $\lambda_{\boldsymbol{A}} = 0.25, \lambda_{\bar{\varphi}} = 0$ without further tuning.

Table 3.2 shows the 1-NN classification results. Again, the Alignment distance can significantly improve the success rate of SoBs, in comparison to the Martin and Max SV distance. This confirms again the advantage of the weighting parameters that allows for prioritizing certain aspects of the KLDS's.

Allover, the proposed framework does not outperform st-TCoF that is based on features extracted by means of trained *Convolutional neural networks* (CNN). This may not be surprising, as CNNs have demonstrated superior recognition performance for other types of visual data. Hence, SoBs based on features learned by CNNs might significantly improve the performance of the presented approach.

By contrast, the proposed approach performs very well in comparison to "classical" methods not based on deep learning, including *Spatiotemporal Oriented En-*

ergy(SOE) [34], *Tree-Structured Vector Quantization* (TSVQ) [89] and *Bag of System Trees* (BoST) [87]. Additionally, learning a BoW codebook can be achieved with little amount of data compared to typical deep learning algorithms.

3.5. Discussion

This chapter discusses the problem of classifying certain types of visual processes. To this end, the observed sequences are converted to streams of histograms and described by KLDS's. Such a model is referred to as a *System of Bags* (SoB). The main contribution is to adapt the framework of the Alignment distance to the case of KLDS's and to apply it for k-NN and NCC classification of SoBs created from dynamic textures and dynamic scenes.

By taking into account the experimental results, we can conclude that the proposed framework has two strengths. Its theoretical motivation is plausible and it performs well both in ablation studies, where the Alignment distance is replaced by other common dissimilarity measures, as well as in comparison to other "shallow" techniques that do not require extensive learning on additional data, such as methods based on sparse dictionaries or wavelet decompositions.

Unfortunately, it cannot be overlooked that more recent deep learning based methods would likely produce significantly better results. In defense of the proposed framework, one may argue that it is only as good as the features collected from the video. For instance, replacing the SURF features that have been used for the YUPENN experiment by descriptors generated by a pre-trained CNN could probably improve performance. The following chapter thus builds upon the approach discussed in the preceding sections, to develop a more effective feature extraction and similarity measure framework for the classification of dynamic textures.

Nevertheless, since such approaches still separate the feature extraction step from the distance-based classification, chances are they still would fail in comparison to state-of-the-art end-to-end methods in classical supervised learning scenarios. On the other hand, with the emergence of *meta-learning* [7] and *few-shot learning* [97] as well as the renaissance of *metric learning* [65], frameworks that separate the feature extraction from the similarity measurement in the classification procedure have experienced some sort of revival, as they make it easier to adapt to changes of data distribution. A potential few-shot learning scenario in future research, for instance, could be to generate the SoBs by means of features from a CNN pre-trained on the meta-training data, and to compute the class centers from the few-shot training samples.

4. Nuclear Distances on Scattering Distributions

While the framework presented in the previous chapter provides a theoretically sound and versatile method for visual process recognition, it offers some room for improvement in terms of performance on recognition datasets, and computational demands.

First, it is obvious that a SoB representation is only as good as the features used to describe the video frames. For instance, the LBP representation used in the dynamic texture experiment may not be the most efficient way to represent textures for recognition tasks and another feature representation may lead to a better performance of the overall framework. Fortunately, designing texture descriptors has been a field of interest in image processing for several decades. It is thus reasonable to assume that it is possible to find more suitable representations.

Second, computation of the Alignment distance is a considerable bottleneck in terms of execution time within the whole recognition process, which is of course due to the optimization on the orthogonal group in Algorithm 7 and similar algorithms. Certain applications, for instance distance-based clustering, require repeated measurement of distance such that a computationally less demanding metric is desirable. Another disadvantage of the Alignment is its requirement of tuning parameters. Since the choice of such parameters always entails some form of engineering, a metric without such a requirement is generally preferable.

Finally, KLDS representations demand a lot of computational resources for storage and processing. Chapter 3 discusses this issue in the context of Fréchet mean computation and proposes to mitigate it by means of a clustering-based Nyström interpolation step. However, the problem already arises earlier, when the KLDS parameters are computed from a video sequence, and the whole set of extracted histograms needs to be stored in order to apply the kernel trick. Applying a similar interpolation technique during the computation of KLDS parameters could reduce their demand for resources.

This chapter proposes a framework for recognition of visual processes that overcomes these challenges. The *Scattering* transform [81], a hierarchical non-linear signal representation based on a repeated application of wavelet filter banks, is used to construct still-image features of texture images which are then used to construct visual process descriptors. Therefore, the focus of this chapter is exclusively on dynamic textures, as opposed to Chapter 3 that also discusses dynamic scenes.

It revises the framework presented in Chapter 3 and makes the following modifications.

- In Chapter 3, dynamic textures are first converted to sequences of LBP histograms prior to subjecting the results to a KLDS parameter extraction algorithm. To do so, a χ^2 kernel as defined in Eq. (3.67) is used. This chapter proposes an approach in which video frames of the dynamic textures are represented by histograms of *Scattering* coefficients. As opposed to Chapter 3, a probability product kernel is employed. This approach is based on insights from [104], where such a feature extraction method yields competitive results in still-image texture retrieval.
- To deal with the computational burden imposed by computing the similarity measure, a simplified version of the Alignment distance is employed. This version neglects the dynamics of the visual process and focuses entirely on its appearance. While such a measure may not as accurately as the Alignment distance account for the dynamic properties of visual processes, it does not require a tedious optimization procedure for computation.
- The Nyström interpolation technique that is employed for the Fréchet mean computation in Chapter 3 is used for the feature extraction of every visual process. Apart from reducing memory usage, this step also speeds up distance measure computation, as smaller matrices are required for the kernel representation of the observation matrix. Hence, it also increases the computational efficiency of approximating Fréchet means.

These adaptations result in a framework based on what will be called *Scattering histogram kernel subspace* (SHKS), in the following. SHKS's yield competitive results for 1-NN classifcation and state-of-the-art results for NCC classifcation of dynamic textures.

4.1. The Scattering Transform

The success of CNNs is often credited to their capability of training on big data sets, which allows for highly parameterizable models. Just as important as data, however, is their *inductive bias* that produces invariances to exactly the type of actions on an image that does not alter its semantic content. For instance, CNNs are known to learn functions that are quite robust to spatial translations. This is particularly desirable for recognition tasks, because changing the spatial location of an object does not alter the object itself. A neural network trained for classifying objects should thus be invariant with respect to the geometrical position of the object to be classified. Similarly, depending on the settings of the camera, the object depiction may exhibit

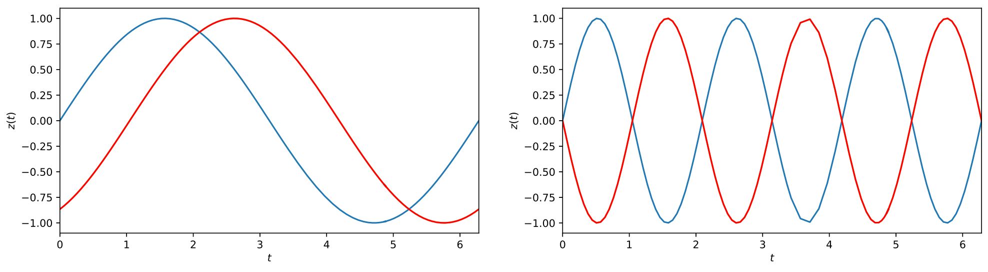


Figure 4.1.: Impact of phase shift on signals of different frequency spectra. The red curve depicts a shift in the time domain of the blue curve by $\tau = \pi/3$. While this leads to a slight perturbation for the slow-changing sine wave with $\omega = 1$ (left), it results in flipping the sign of a fast-changing wave with $\omega = 3$ (right).

different types of spatial deformations that should also not perturb a classification architecture.

Scattering transforms are an attempt to reproduce this inductive bias without training. In essence, a Scattering transform is a CNN with fixed weights and without channel recombination, and in which the absolute value operator is consistently used as the activation function. We can motivate Scattering transforms by the observation that processing a signal with a low-pass filter $\bar{\psi}$ makes it less sensitive to local deformations and shift operations. This can be easily illustrated by considering the integrated squared difference between a 1D signal $z(t) = \sin(\omega t)$ and a translated version. If the frequency ω is high, the squared error

$$\int_0^{2\pi} |z(t) - z(t-\tau)|^2 \,\mathrm{d}\, t \tag{4.1}$$

introduced due to a small shift by τ (in relation to ω) is considerably higher than for small ω. Figure 4.1 visualizes this phenomenon.

At the same time, a lowpass filter causes a loss of information, in particular for natural images. The lost information can be retained by not only collecting the low-frequency bands of an image, but also the bands belonging to higher-frequencies. This can be obtained using a set of band-pass filters, where each one of the filters covers a different region of the spectral domain. Such filters are often created by using a *mother wavelet* that is then scaled and rotated with different factors and angles in the spacial domain resulting in a shift-equivariant *wavelet frame* [80]. A wavelet frame constructed in such a way can be used as a filter bank to create a decomposition of the input signal into several band-limited subbands. With increasing frequency, these subbands become less and less invariant to translations, deformations and similar transformations of the input image, because the effect of these actions is retained in the higher-frequency subbands. Thus, signal representations based on subband decompositions require a trade-off between robustness and descriptiveness of signal representations produced. On the one hand, the high frequencies contained in these subbands pose considerable robustness problems due to their deformation sensitivity. On the other hand, simply abandoning higher-frequency subbands would lead to a loss of information, because abrupt changes such as edges or corners that are crucial in recognition tasks [77] are mostly contained in the high frequencies. Since recognition tasks typically require representations that are both robust and descriptive, it would be desirable to get rid of the higher frequencies without losing the information contained in them. As it turns out, this can be obtained by applying the absolute value operator to each subband coefficient. By eliminating changes of sign, the absolute value reduces the share of high frequencies in a signal. Moreover, it does so without removing too much relevant information from the signal representation as the sign is rarely significant for recognition tasks and can be often retained using phase-retrieval algorithms [43].

Let $\psi_1, \dots, \psi_J$ be a set of bandpass filters corresponding to different regions of the spectral domain, and let L^2 refer to the Lebesgue space of square-integrable functions. Consider the operator

$$\begin{aligned} \Psi^J : L^2 &\to \underbrace{L^2 \times \cdots \times L^2}_{J+1 \text{ times}}, \\ z &\mapsto \left(\bar{\psi} * z, \quad |\psi_1 * z|, \quad \cdots \quad |\psi_J * z|\right), \end{aligned} \tag{4.2}$$

where the asterisk $*$ describes a convolution of two signals. In words, Ψ^J creates a subband decomposition of the input, where the absolute value operator is applied to each non-lowpass output. The first element of the output tuple, written as $\Psi^J[z]_0$, is a lowpass representation of the input and possesses thus the robustness properties of low-frequency signals that have been discussed above. The other signals $\Psi^J[z]_1, \dots, \Psi^J[z]_J$ contained in the tuple can be subjected to such an operator repeatedly, yielding another J subband decompositions. This procedure can be repeated several times, yielding a hierarchical tree structure like in Figure 4.2, where the black circles denote the absolute values of bandpass signals and the white circles denote lowpass signals. With each new layer, information from the input image finds its way into the lowpass outputs of Ψ^J, due to the elimination of sign changes. After a few layers, a robust, yet descriptive representation of the input is generated.

The M-depth Scattering transform of a signal z is the collection of the lowpass signals created by constructing such a tree with M layers, and collecting all the white circles from it. We refer to these lowpass signals as the Scattering *subbands*. Each Scattering subband of z in a layer $m \in \{1, \dots, M-1\}$ is identified by the filter indexes $j_m \in \{1, \dots, J_m\}$ and can be written as

$$\begin{aligned} \mathcal{S}_{j_1,\dots,j_m}[z] =& \Psi^{J_{m+1}}[\Psi^{J_m}[\cdots[\Psi^{J_1}[z]_{j_1}]\cdots]_{j_m}]_0 \\ =& \bar{\psi} * |\psi_{j_m} * |\cdots * |\psi_{j_1} * |z||\cdots||, \end{aligned} \tag{4.3}$$

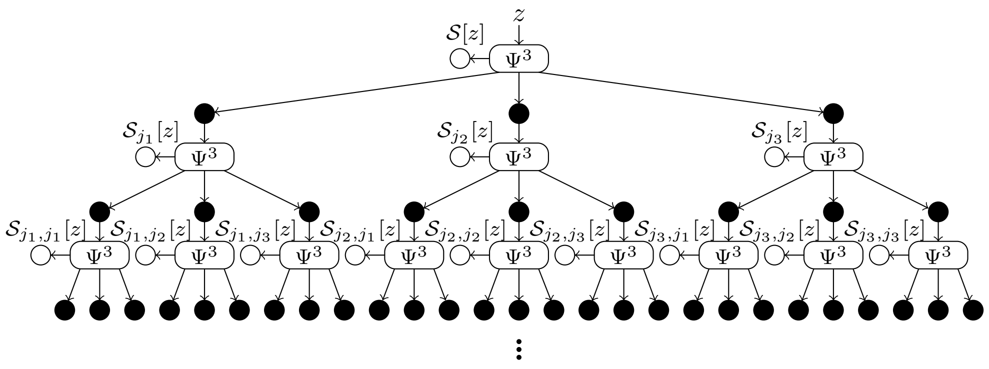

z
$\mathcal{S}[z]$
Ψ^3
$\mathcal{S}_{j_1}[z]$
$\mathcal{S}_{j_2}[z]$
$\mathcal{S}_{j_3}[z]$
$\mathcal{S}_{j_1,j_1}[z]$
$\mathcal{S}_{j_1,j_2}[z]$
$\mathcal{S}_{j_1,j_3}[z]$
$\mathcal{S}_{j_2,j_1}[z]$
$\mathcal{S}_{j_2,j_2}[z]$
$\mathcal{S}_{j_2,j_3}[z]$
$\mathcal{S}_{j_3,j_1}[z]$
$\mathcal{S}_{j_3,j_2}[z]$
$\mathcal{S}_{j_3,j_3}[z]$

Figure 4.2.: Scattering tree produced by successive application of Ψ^3 on the input signal z. Lowpass signals are depicted as white nodes. The absolute values of bandpass signals are depicted as black nodes. Once the tree is computed, only the white nodes are kept as a representation of the input signal.

Therefore, the Scattering transform of z is a tuple containing all

$$N_{\text{Bands}} = 1 + \sum_{m=1}^{M-1} \prod_{i=1}^{m} J_i \tag{4.4}$$

subbands. For the subband at input input layer ($m = 0$), let us fix the notation

$$\mathcal{S}[z] = \bar{\psi} * z. \tag{4.5}$$

As an optional step, the Scattering subbands can be normalized in order to decorrelate them. The procedure to do so has been originally proposed in [4]. Let us define the *normalized Scattering transform* following the proposal in [4], namely as a collection of subbands $\tilde{\mathcal{S}}_{j_1,\ldots,j_m}$ that can be computed from the regular Scattering subbands $\mathcal{S}_{j_1,\ldots,j_m}$ of a signal z. In the normalized Scattering transform, the input layer ($m = 0$) is identical to the regular Scattering transform, i.e.,

$$\tilde{\mathcal{S}}[z] = \mathcal{S}[z], \tag{4.6}$$

the first layer ($m = 1$) is normalized by the signal average of the input, denoted by avg, i.e.,

$$\tilde{\mathcal{S}}_{j_1}[z] = \frac{\mathcal{S}_{j_1}[z]}{\text{avg}(z)}, \tag{4.7}$$

and all remaining layers ($m > 1$) are normalized using the parent node, i.e.,

$$\tilde{\mathcal{S}}_{j_1,\ldots,j_m}[z] = \frac{\mathcal{S}_{j_1,\ldots,j_m}[z]}{\mathcal{S}_{j_1,\ldots,j_{m-1}}[z]}. \tag{4.8}$$

It has been observed that normalizing the Scattering coefficients using this scheme can improve recognition for texture images and voice recordnings [4, 104].

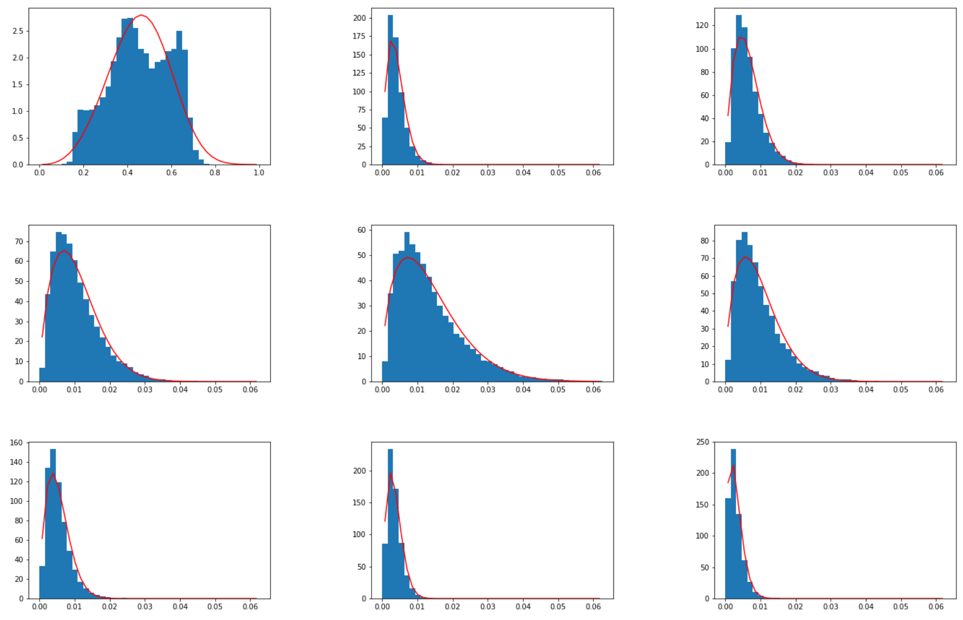


Figure 4.3.: Coefficient distributions (blue: histograms, red: Weibull fittings) of different Scattering subbands

4.2. Scattering Distributions of Texture Images

Statistical moments of multi-scale subband representations are known to be an expressive descriptor of texture images in recognition and retrieval tasks [36, 67, 70]. In [104], a texture retrieval method based on Scattering subband distributions has been proposed. Texture images are first subjected to a Scattering transform and a parameterized model for the coefficient distribution in each Scattering subband is computed. Specifically, a two-parameters *Weibull* distribution [92] is used as a statistical model. The two Weibull parameters are approximated using maximum likelihood estimation [114] on each Scattering subband individually.

A visualization of this idea is given in Fig. 4.3, where Scattering subband histograms are compared to the estimated Weibull curve. Since each of these curves is described by two parameters, the resulting feature vectors describing the texture images contain $2N_{\mathrm{Bands}}$ elements. To define a distance measure, an approximation of the *Bhattacharyya* kernel, defined in [60] for two probability distributions p_1, p_2 over a space Ω as

$$\kappa_{\mathrm{Bht}}(p_1, p_2) = \int_{\Omega} \sqrt{p_1(\boldsymbol{\xi}) p_2(\boldsymbol{\xi})}\, \mathrm{d}\, \boldsymbol{\xi}, \tag{4.9}$$

is used. This approximation is directly computed from the Weibull parameter based feature vectors. Note that for independent multivariate distributions, the Bhattacharyya kernel can be computed as the product over all marginal distributions.

The framework presented in the following is based on the same feature extraction mechanism, but instead of fitting a continuous parameterized distribution to the subband coefficients, the distributions are directly modeled as histograms. This improves accuracy of the distribution approximations and reduces the time used for extracting the features since computing a histogram is typically faster than estimating Weibull parameters using maximum likelihood. The downside is that computing the Bhattacharryya kernel takes longer. However, it can be computed exactly rather than using an approximated version as it is done in [104].

4.3. Kernel Subspaces of Scattering Histograms

The feature extraction algorithm employed in this chapter works as follows. First, a dynamic texture video

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1 & \cdots & \boldsymbol{z}_N \end{bmatrix} \tag{4.10}$$

is converted to a sequence of feature vectors by applying the Scattering transform, written as ST, to each frame. Every subband is then used to compute a histogram from its coefficients. This results in a matrix $\boldsymbol{Y} \in \mathbb{R}^{N_{\mathrm{Bands}} N_{\mathrm{bins}} \times N}$, where N is the sequence length, N_{Bands} is the number of Scattering subbands and N_{bins} the number of histogram bins.

Let us assume that the resulting feature vectors, i.e., the columns of $\boldsymbol{Y}$, are ordered in a way such that the jth bin of the histogram belonging to the ith Scattering subband is given by $i \cdot N_{\mathrm{Bins}} + j$th element of the vector. Note that we start counting the subbands from 0. The Bhattacharyya kernel between the histograms of ith subband of two feature vectors $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^{N_{\mathrm{Bands}} N_{\mathrm{bins}}}$ is given by the sum of the element-wise products, after the square root was applied to them, i.e.,

$$\sum_{j=1}^{N_{\mathrm{bins}}} \sqrt{(\boldsymbol{y}_1)_{i \cdot N_{\mathrm{bins}}+j} \cdot (\boldsymbol{y}_2)_{i \cdot N_{\mathrm{bins}}+j}}. \tag{4.11}$$

By assuming that the subbands are independent, we can define the Bhattacharyya kernel on a pair of two feature vectors $\boldsymbol{y}_1, \boldsymbol{y}_2$ as the product over all subbands, i.e.,

$$\begin{aligned} \kappa_{\mathrm{Bht}} \colon \mathbb{R}^{2 \cdot N_{\mathrm{Bands}} N_{\mathrm{bins}}} &\to \mathbb{R}, \\ \boldsymbol{y}_1, \boldsymbol{y}_2 &\mapsto \prod_{i=0}^{N_{\mathrm{Bands}}-1} \sum_{j=1}^{N_{\mathrm{bins}}} \sqrt{(\boldsymbol{y}_1)_{i \cdot N_{\mathrm{bins}}+j} \cdot (\boldsymbol{y}_2)_{i \cdot N_{\mathrm{bins}}+j}}. \end{aligned} \tag{4.12}$$

Given the representation $\boldsymbol{Y}$ and kernel κ_{Bht}, we could proceed to compute a KLDS using these defintions, as it was done in Chapter 3. however, we aim for a

Algorithm 9 : Scattering histogram kernel subspace computation

```
Input : Video Sequence $\boldsymbol{Z} = \begin{bmatrix}\boldsymbol{z}_1 & \cdots & \boldsymbol{z}_N\end{bmatrix}$, Histogram size $N_{\mathrm{Bins}} \in \mathbb{N}$,
        Sampling size $\tilde{N} \geq n$, latent dimension $n \in \mathbb{N}$
1 $\boldsymbol{S} \leftarrow \mathrm{ST}(\boldsymbol{Z})$;            // Frame-wise Scattering transform
2 $\boldsymbol{Y} \leftarrow \mathrm{hist}(\boldsymbol{S}, N_{\mathrm{Bins}})$ ; // Subband-wise histogram computation
3 $\tilde{\boldsymbol{Y}} \leftarrow \mathrm{SMPL}_{\tilde{N}}(\boldsymbol{Y})$ ;                              // Subsampling
4 $\boldsymbol{U}, \boldsymbol{\Lambda}, \boldsymbol{U} \leftarrow \mathrm{SVD}(\kappa_{\mathrm{Bht}}(\boldsymbol{Y}, \boldsymbol{Y}))$;
5 $\boldsymbol{R} \leftarrow \boldsymbol{U}_n {\boldsymbol{\Lambda}_{(n)}^{(n)}}^{-1/2}$;
6 $\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{\Lambda}}, \tilde{\boldsymbol{U}} \leftarrow \mathrm{SVD}(\kappa_{\mathrm{Bht}}(\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{Y}}))$;
7 $\boldsymbol{U}', \_, \boldsymbol{V}' \leftarrow \mathrm{SVD}(\boldsymbol{R}^{\top} \kappa_{\mathrm{Bht}}(\boldsymbol{Y}, \tilde{\boldsymbol{Y}}) \tilde{\boldsymbol{U}} \tilde{\boldsymbol{\Lambda}}^{-1/2})$;
8 $\tilde{\boldsymbol{R}} \leftarrow \tilde{\boldsymbol{U}} \tilde{\boldsymbol{\Lambda}}^{-1/2} \boldsymbol{V}' \boldsymbol{U}'^{\top}$;                  // Nyström interpolation
Output : Kernel Subspace parameters $\tilde{\boldsymbol{R}}, \tilde{\boldsymbol{Y}}$
```

slightly simpler approach here. The first adaptation we make is to set $\bar{\varphi} = 0$ in Eq. (3.3). That way, the model becomes less accurate, but we avoid modeling the affine subspace offset as a separate parameter $\boldsymbol{\beta}$. Next, recall Figure 3.4. It can be observed, that the weighting parameter $\lambda_{\boldsymbol{A}}$ has little to no impact on the recognition performance. More generally, it is safe to say that the dynamic parameters $\boldsymbol{A}$ and $\boldsymbol{B}$ are hardly of any relevance for the recognition of dynamic textures modeled as KLDS's of histograms, because recognition can be carried out based on the appearance of individual frames. Neglecting the parameters $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{\beta}$ reduces Algorithm 6 to only computing $\boldsymbol{R}$.

An additional adaptation to Algorithm 6 that can help reduce computational demands is the Nyström interpolation of the input sequences. Specifically, we can compress the extracted features using a Nyström-type interpolation technique resembling the one in Chapter 3, by subsampling the feature sequence $\boldsymbol{Y}$. Let us denote this sampling operation by

$$\begin{aligned} \mathrm{SMPL}_{\tilde{N}} &: \mathbb{R}^{d \times N} \to \mathbb{R}^{d \times \tilde{N}} \\ \begin{bmatrix}\boldsymbol{y}_1 & \cdots & \boldsymbol{y}_N\end{bmatrix} &\mapsto \begin{bmatrix}\boldsymbol{y}_1 & \boldsymbol{y}_{1+\lfloor N/\tilde{N} \rfloor} & \cdots & \boldsymbol{y}_{1+(\tilde{N}-1)\lfloor N/\tilde{N} \rfloor}\end{bmatrix}. \end{aligned} \tag{4.13}$$

The operator $\mathrm{SMPL}_{\tilde{N}}$ is applied to gather a representative submatrix $\tilde{\boldsymbol{Y}}$ of $\boldsymbol{Y}$. This submatrix contains significantly fewer columns than the original feature matrix $\boldsymbol{Y}$ and is later used to construct a subspace of the kernel feature space. This resembles line 1 of Algorithm 8, but instead of using a clustering algorithm, the columns of $\boldsymbol{Y}$ are directly gathered using equidistant sampling of the video sequence.

Altogether, we end up with Algorithm 9. The returned parameters $\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{R}}$ describe a subspace that approximates the space described by the full matrices $\boldsymbol{Y}, \boldsymbol{R}$. More precisely, parameter $\tilde{\boldsymbol{R}}$ is chosen such that the basis of said subspace is orthonormal, i.e., the equality

$$\tilde{\boldsymbol{R}}^\top \kappa_{\mathrm{Bht}}(\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{Y}})\tilde{\boldsymbol{R}} = \boldsymbol{I}_n \tag{4.14}$$

holds, and an appropriate distance measure to the parameter pair $\boldsymbol{Y}, \boldsymbol{R}$ is minimized. The most obvious choice for such a distance is the sum of squared metric distances between the respective basis vectors in the kernel feature space. We can write this distance for two subspace parameter pairs $(\boldsymbol{R}_1, \boldsymbol{Y}_1)$ and $(\boldsymbol{R}_2, \boldsymbol{Y}_2)$ as

$$\begin{aligned} d_{\mathrm{se}}^2((\boldsymbol{R}_1, \boldsymbol{Y}_1), (\boldsymbol{R}_2, \boldsymbol{Y}_2)) = &\operatorname{tr}(\boldsymbol{R}_1{}^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_1)\boldsymbol{R}_1) - 2\operatorname{tr}(\boldsymbol{R}_1{}^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2)\boldsymbol{R}_2) \\ &+ \operatorname{tr}(\boldsymbol{R}_2{}^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_2, \boldsymbol{Y}_2)\boldsymbol{R}_2). \end{aligned} \tag{4.15}$$

The solution for $\tilde{\boldsymbol{R}}$ can be thus written as

$$\begin{aligned} \tilde{\boldsymbol{R}} &= \underset{\boldsymbol{R}' \text{ s.t. } \boldsymbol{R}'^\top \kappa_{\mathrm{Bht}}(\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{Y}})\boldsymbol{R}' = \boldsymbol{I}_n}{\arg\min} d_{\mathrm{se}}^2((\boldsymbol{R}', \tilde{\boldsymbol{Y}}), (\boldsymbol{R}, \boldsymbol{Y})) \\ &= \underset{\boldsymbol{R}' \text{ s.t. } \boldsymbol{R}'^\top \kappa_{\mathrm{Bht}}(\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{Y}})\boldsymbol{R}' = \boldsymbol{I}_n}{\arg\max} \operatorname{tr}(\boldsymbol{R}'^\top \kappa_{\mathrm{Bht}}(\tilde{\boldsymbol{Y}}, \boldsymbol{Y})\boldsymbol{R}). \end{aligned} \tag{4.16}$$

Analogously to Chapter 3, this problem can be solved using the SVD of $\kappa(\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{Y}})$ and an additional factor matrix with orthogonal columns that maximizes the trace product. In Algorithm 9, this step is carried out in lines 6-8.

4.4. The Nuclear Distance

Like the LDS and KLDS parameter tuples introduced in Chapter 2 and Chapter 3, respectively, the parameter pairs returned by Algorithm 9 are not unique with regards to the kernel feature subspace they represent. Specifically, an orthogonal transformation of the matrix $\tilde{\boldsymbol{R}}$ yields a new parameter pair that describes the same subspace using a different basis. In defining a distance measure, it is thus again sensible to demand that the distance is invariant to such changes of basis in the kernel feature space. Given two parameter pairs $(\boldsymbol{R}_1, \boldsymbol{Y}_1)$ and $(\boldsymbol{R}_2, \boldsymbol{Y}_2)$, a natural choice of such a distance can be obtained by computing Eq. (4.15) for the orthogonal basis transformation that minimizes d_{se}^2, i.e.,

$$d_{\mathrm{ncl}}^2((\boldsymbol{R}_1, \boldsymbol{Y}_1), (\boldsymbol{R}_2, \boldsymbol{Y}_2)) = \min_{\boldsymbol{Q} \in \mathrm{O}(n)} d_{\mathrm{se}}^2((\boldsymbol{R}_1, \boldsymbol{Y}_1), (\boldsymbol{R}_2\boldsymbol{Q}^\top, \boldsymbol{Y}_2)). \tag{4.17}$$

Recall that we generally assume that the bases of the kernel feature subspaces described by $(\boldsymbol{R}_1, \boldsymbol{Y}_1)$ and $(\boldsymbol{R}_2, \boldsymbol{Y}_2)$ are orthonormal. Thus, due to the condition

$$\operatorname{tr}(\boldsymbol{R}_1^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_1)\boldsymbol{R}_1) = \operatorname{tr}(\boldsymbol{R}_2^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_2, \boldsymbol{Y}_2)\boldsymbol{R}_2) = \boldsymbol{I}_n, \tag{4.18}$$

Algorithm 10 : Fréchet mean computation

```
   Input : Set of parameter pairs $\{(\boldsymbol{R}_i, \boldsymbol{Y}_i)\}_{i\in\{1,\dots,K\}}$, sampling size $\bar{N} \in \mathbb{N}$
1  $\bar{\boldsymbol{Y}} \leftarrow k-\mathrm{means}\left(\begin{bmatrix}\boldsymbol{Y}_1 & \cdots & \boldsymbol{Y}_K\end{bmatrix}, \bar{N}\right)$;
2  $\boldsymbol{U}, \boldsymbol{\Lambda}, \_ \leftarrow \mathrm{SVD}(\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}}))$;
3  foreach $i \in \{1, \dots, K\}$ do
4      $\boldsymbol{Q}_i \leftarrow \boldsymbol{I}_n$;                                   // Initialization
5  end
6  while $g(\bar{\Xi})$ not converged do
7      $\boldsymbol{U}', \_, \boldsymbol{V}'^\top \leftarrow \mathrm{SVD}\left(\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}} \boldsymbol{U}_{(r)}^\top \sum_{i=1}^K \kappa(\bar{\boldsymbol{Y}}, \boldsymbol{Y}_i)\boldsymbol{R}_i\boldsymbol{Q}_i^\top\right)$;
        // Eq. (3.61), $r$ is rank of $\kappa(\bar{\boldsymbol{Y}}, \bar{\boldsymbol{Y}})$
8      $\bar{\boldsymbol{R}} \leftarrow \boldsymbol{U}_{(r)}\boldsymbol{\Lambda}_{(r)}^{(r)-\frac{1}{2}}\boldsymbol{U}'_{(n)}\boldsymbol{V}'^\top$;                // Eq. (3.62)
9      foreach $i \in \{1, \dots, K\}$ do
10         $\boldsymbol{U}_i, \_, \boldsymbol{V}_i \leftarrow \mathrm{SVD}(\bar{\boldsymbol{R}}^\top \kappa_{\mathrm{Bht}}(\bar{\boldsymbol{Y}}, \boldsymbol{Y}_i)\boldsymbol{R}_i)$;
11         $\boldsymbol{Q}_i \leftarrow \boldsymbol{V}_i\boldsymbol{U}_i^\top$;
12     end
13 end
   Output : Estimated Fréchet mean $\bar{\boldsymbol{R}}, \bar{\boldsymbol{Y}}$
```

we can rewrite Eq. (4.17) as

$$\begin{aligned} d_{\mathrm{ncl}}^2((\boldsymbol{R}_1, \boldsymbol{Y}_1), (\boldsymbol{R}_2, \boldsymbol{Y}_2)) &= 2n - 2 \max_{\boldsymbol{Q}\in O(n)} \mathrm{tr}(\boldsymbol{R}_1^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2)\boldsymbol{R}_2\boldsymbol{Q}^\top) \\ &= 2n - 2\|\boldsymbol{R}_1^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2)\boldsymbol{R}_2\|_*. \end{aligned} \tag{4.19}$$

We call Eq. (4.19) the *Nuclear distance* due to the nuclear norm applied to $\boldsymbol{R}_1^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2)\boldsymbol{R}_2$. It can be alternatively defined as the Alignment distance $d_{\mathrm{align},\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}},\lambda_{\bar{\varphi}}}$ with $\lambda_{\boldsymbol{A}} = \lambda_{\boldsymbol{B}} = \lambda_\varphi = 0$. However, computation of Eq. (4.19) is much less time-costly than optimizing over $\mathrm{O}(n)$ in order to compute the Alignment distance, as it suffices to retrieve the singular values of $\boldsymbol{R}_1^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2)\boldsymbol{R}_2$ to compute the nuclear norm.

Likewise, computing the Fréchet mean with respect to d_{ncl} is simpler than computing it with respect to the Alignment distance, since the parameters $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{\beta}$ are neglected, and minimizing for $\boldsymbol{Q}_i$ can be performed via the SVD. With

$$g_{\mathrm{ncl}}(\boldsymbol{R}, \boldsymbol{Y}) = \sum_{i=1}^K d_{\mathrm{ncl}}^2((\boldsymbol{R}, \boldsymbol{Y}), (\boldsymbol{R}_i, \boldsymbol{Y}_i)), \tag{4.20}$$

the Fréchet mean is given by

$$\begin{aligned} \bar{\boldsymbol{R}}, \bar{\boldsymbol{Y}} &= \arg\min_{\boldsymbol{R},\boldsymbol{Y}} g_{\mathrm{ncl}}(\boldsymbol{R}, \boldsymbol{Y}), \\ &\quad \text{s.t. } \boldsymbol{R}^\top \kappa_{\mathrm{Bht}}(\boldsymbol{Y}, \boldsymbol{Y})\boldsymbol{R} = \boldsymbol{I}_n. \end{aligned} \tag{4.21}$$

The procedure to solve the optimization problem in Eq. (4.21) is described in Algorithm 10. Like Algorithm 8, it is based on an alternating approach, in which every iteration contains a **foreach** loop to find a set of matrices $\boldsymbol{Q}_1, \dots \boldsymbol{Q}_K \in \mathrm{O}(n)$, such that

$$d^2_{\mathrm{ncl}}((\boldsymbol{R}, \boldsymbol{Y}), (\boldsymbol{R}_i, \boldsymbol{Y}_i)) = d^2_{\mathrm{se}}((\boldsymbol{R}, \boldsymbol{Y}), (\boldsymbol{R}_i \boldsymbol{Q}_i^\top, \boldsymbol{Y}_i)) \tag{4.22}$$

is satisfied for all $i \in \{1, \dots, K\}$. This allows us to use d^2_{se} instead of d^2_{ncl} in Eq. (4.20). By writing Eq. (4.20) as

$$g_{\mathrm{ncl}}(\boldsymbol{R}, \boldsymbol{Y}) = \sum_{i=1}^{K} d^2_{\mathrm{se}}((\boldsymbol{R}, \boldsymbol{Y}), (\boldsymbol{R}_i \boldsymbol{Q}_i^\top, \boldsymbol{Y}_i)), \tag{4.23}$$

an approximation of $\bar{\boldsymbol{R}}$ can be computed that minimizes Eq. (4.23) for $\boldsymbol{R}$. This is done using the SVD at the beginning of each iteration. The parameter $\bar{\boldsymbol{Y}}$ is estimated by performing a Nyström interpolation of $\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_K$, which has also been already done in Algorithm 8..

4.5. Experiments

The following experiment repeats the DynTex evaluation from Chapter 3 with the proposed *Scattering histogram kernel subspaces* (SHKS) and *normalized KSHS* (NKSHS), where *normalization* refers to the operations on the Scattering subbands described in Section 4.1. The experiments were implemented in Python. Each video was converted to grayscale, and transformed via Kymatio [6] with the parameters `J=4, L=4` to compute the Scattering coefficients. For each Scattering subband, a histogram with $N_{\mathrm{Bins}} = 20$ bins has been computed. (N)KSHS parameters have been computed via Algorithm 9 with $\tilde{N} = 15$ and $n = 5$.

Table 4.1 shows the results in comparison to the baseline methods in Chapter 3. In accordance with the results in [104], normalizing the Scattering subbands improves the recognition performance. While the reason for this is not entirely clear, one can speculate that this has to do with the decorrelating effect of normalizing the Scattering coefficients [4]. The coefficients of different Scattering subbands without normalization tend to be strongly correlated. Normalizing the coefficients according to the scheme discussed in Section 4.1 mitigates this phenomenon such that it becomes harder to predict coefficients in one subband from the coefficients in another one. The assumption of independent distributions becomes therefore more realistic, which justifies the computation of Bhattacharyya kernel using the product formula stated in Eq. (4.12).

Overall, the results compete quite well with other state-of-the-art approaches in terms of classification rate. While SOE-NET [48] and STRF N-jet [59] yield slightly better results for 1-NN, NCC classification via Fréchet means computed with the iterative averaging procedure in Algorithm 10 consistently produces the best results out of all listed methods.

	Alpha		Beta		Gamma	
	1-NN	NCC	1-NN	NCC	1-NN	NCC
LBP-TOP	96.7 %	-	85.8 %	-	84.9 %	-
ASF-TOP	91.7 %	-	86.4 %	-	89.4 %	-
MBSIF-TOP	90.0 %	-	90.7 %	-	-*	-
PCANet-Top	96.7 %	-	90.7 %	-	89.4 %	-
DFS	-	83.6 %	-	65.2 %	-	60.8 %
2D+T Curvelet	-	85.0 %	-	67.0 %	-	-*
OTDL	-	86.6 %	-	69.0 %	-	64.2 %
CLSP-TOP	95.0 %	-	92.0 %	-	**91.3** %	-
STRF N-jet	**100.0 %**	-	93.8%	-	91.2 %	-
B3DF	96.7 %	90.0 %	90.1 %	74.1 %	-*	-*
SOE-NET	98.3 %	**96.7 %**	**96.9 %**	86.4 %	-*	-*
SoB + Align	98.3 %	88.3 %	90.1 %	75.3 %	79.9 %	67.1 %
KSHS+Ncl.	98.3 %	**96.7 %**	88.9 %	88.3 %	88.6 %	86.7 %
KNSHS+Ncl.	98.3 %	**96.7 %**	93.2 %	**90.1 %**	**91.3** %	**89.8 %**

*Reported results refer to 275-video version of DynTex Gamma.

Table 4.1.: Recognition rate on DynTex subsets

4.6. Discussion

This chapter combines theoretical and experimental insights from Chapter 3 with results on content-based still-image texture retrieval to develop a dynamic texture recognition framework. It employs the Scattering transform to convert video frames to robust and expressive representations which are then converted into vectors of concatenated histograms. These vectors are then subjected to a Bhattacharyya kernel that is used to compute kernelized subspace representations of the dynamic texture sequences. For recognition purposes, a distance measure is employed that can be inferred from the Alignment distance by neglecting the state space dynamics of the visual processes. This yields a metric called the Nuclear distance which requires less time for computation than the original Alignment distance. Furthermore, it eliminates the risk of local minima during distance computation, since the distance does not need to be computed via optimization over the orthogonal group, e.g. by using Algorithm 7. Instead, it can be obtained analytically by means of basic matrix algebra.

The evaluation on the DynTex dataset shows a significant improvement over SoB+Align, based on LBP features, as well as over many other state-of-the-art algorithms. This is particularly true for NCC classification, for which the proposed approach outperforms a large number of techniques discussed in literature. The

NCC performance also indicates applicability to unsupervised learning tasks that are based on averaging operations, such as clustering.

Like the framework presented in Chapter 3, this chapter discusses a technique based on kernel PCA. One issue with kernel PCA-based techniques is the lack of any straight-forward way to reconstruct samples in $\mathbb{R}^d$ from their representation vectors in $\mathbb{R}^n$. In principle, such reconstructions can be computed iteratively by minimizing a kernel based error expression [85]. Practically, this is computationally too demanding to perform in real-time. As a consequence, KLDS's (and KSHS's) cannot be easily incorporated into applications such as synthesis or anomaly detection, because it is very difficult to infer statistical implications on trajectories in the observation space from them. Applications that require the possibility to reconstruct observations from latent state space variables are thus a strong argument against kernel-based methods.

Hence, the remainder of this thesis focuses on the “generative” aspect of visual process modeling. That is to say that the question of how to practically generate video frames from latent space representations becomes of integral relevance for the ongoing discussion. This is most conveniently achieved using deep learning, as it has been demonstrated before that CNNs are capable of generating photo-realistic, yet synthetic, images [99]. In the following chapter, a CNN based approach that is capable of generative modeling of visual processes is presented.

5. Dynamic Variational Autoencoders

Chapter 3 and Chapter 4 discuss ways to represent video sequences by trajectories in a lower-dimensional latent state space using kernels. Since there is no simple way to reconstruct high-dimensional data points from their kernel PCA representations, the framework is ill-suited for converse tasks that require to infer insights about the behaviour in the observation space from corresponding latent space trajectories. For feature-based tasks such as classification or clustering, this is not necessary. If, however, the aim is not to just represent visual processes, but to predict future frames from previous ones, compute marginal frame distributions for one or several time steps, or generate new sequences, it is of great help to have an explicit representation of the observation function Γ. Such a representation turns Eq. (1.6) into a simple and yet powerful model for the visual process at hand. Consider for instance the problem of observing a sequence of T frames and evaluating if, according to the learned model, such a sequence is likely to occur, i.e., if the probability according to the mode of such a sequence of frames is high or not. This could be done by sampling a large number of VAR sequences with length T, transforming them to the observational space via Γ, and estimating the likelihood of the observed sequence based on a distribution model inferred from the generated sequences. Likewise, the problem of frame prediction becomes also considerably simple. Given an observed frame $\boldsymbol{y}_t$, predicting the follow-up frame $\boldsymbol{y}_{t+1}$ could be done by first estimating the according latent state $\boldsymbol{x}_t$ and projecting it one step into the future by means of the state transition matrix $\boldsymbol{A}$, prior to transforming the result to the observation space by means of Γ as $\tilde{\boldsymbol{y}}_{t+1} = \Gamma(\boldsymbol{A}\boldsymbol{x}_t)$.

For a function Γ to be an appropriate choice for the observer in Eq. (1.6), the set

$$\mathbb{M} = \{\Gamma(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^n\} \subset \mathbb{R}^d \tag{5.1}$$

should be an appropriate model for the observation space of image frames created by the respective visual process. It has been widely observed that CNNs are capable of implementing functions that parameterize sets of real-world images, e.g. in the context of deep generative learning [20, 99] or inverse problem solving [121].

Employing an appropriate CNN as the observation function transforms Eq. (1.6) into

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{v}_t, \tag{5.2a}$$

$$\boldsymbol{y}_t = \Gamma_{\boldsymbol{\zeta}}(\boldsymbol{x}_t), \tag{5.2b}$$

where $\Gamma_{\boldsymbol{\zeta}}$ is a CNN with learnable weights $\boldsymbol{\zeta} \in \mathbb{R}^D$ with D representing the number of learnable parameters.

Let us denote by

$$\mathcal{F} = \left\{ f_{\boldsymbol{\nu}} \colon \mathbb{R}^n \to \mathbb{R}^d \mid \boldsymbol{\nu} \in \mathbb{R}^D \right\}, \tag{5.3}$$

the set of functions that can be realized using a neural network of some architecture with D fixed parameters, e.g. weights and biases. In the following, it will be assumed that the architecture, i.e., the number and breadth of the individual neural network layers, and the choice of additional components such as skip connections or batch normalization layers is fixed, as the choice of architecture of the neural network implementing the observation function $\Gamma_{\boldsymbol{\zeta}}$ is not the focus of this chapter. It will be thus assumed from now on that $\Gamma_{\boldsymbol{\zeta}}$ belongs to the set $\mathcal{F}$.

The choice of an appropriate neural architecture for $\Gamma_{\boldsymbol{\zeta}}$, i.e., the definition of $\mathcal{F}$ can be driven by previous results on image generation. However, the problem of training $\Gamma_{\boldsymbol{\zeta}}$, that is to say, the optimization on $\mathcal{F}$, such that $\Gamma_{\boldsymbol{\zeta}}$ maps VAR noise to plausible visual process trajectories is not always easily solvable. As discussed in Section 2.6, separately learning the latent state space model in Eq. (5.2a) and the observation function of Eq. (5.2b) may not bring the desired results, because it is hard to justify that the optimality gap caused by learning these two aspects separately does not have considerable impact. This chapter develops a method to learn the parameters $\boldsymbol{A}$ and $\boldsymbol{B}$ of Eq. (5.2a) jointly, alongside the observation function $\Gamma_{\boldsymbol{\zeta}}$ of Eq. (5.2b). Specifically, the contributions are as follows.

- A deep generative framework is introduced with the capability to learn models described by Eq. (5.2) from video sequences of visual processes. To this end, a modified VAE is employed, in which the function $\Gamma_{\boldsymbol{\zeta}}$ is realized by its decoder. The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are implemented using an additional linear (dense) neural layer that can be trained alongside the VAE. That way, $\boldsymbol{A}$ and $\boldsymbol{B}$ can be treated as weights of the neural network and thus trained simultaneously with $\Gamma_{\boldsymbol{\zeta}}$.

- A regularization is proposed which ensures that the marginal distribution of the latent space does not change over time. This accounts for stationarity assumptions and allows for stochastic optimization algorithms that make it possible to train the neural architecture by providing only one pair of video frames at a time.

- In order to evaluate the proposed framework, synthesis experiments on synthetic sequences and on dynamic textures are carried out. To this end, the framework is first trained using frame pairs of a training video, and then a synthesized sequence is generated by sampling from autoregressive noise described by $\boldsymbol{A}$ and $\boldsymbol{B}$, before mapping the noise to $\mathbb{M}$ using $\Gamma_{\boldsymbol{\zeta}}$.

It is worth noting that this chapter is not the first approach to combine VAR-like state transitions with VAEs. For instance, one of the works in literature that has done so is the graphical model presented in [62]. It relies on conditional random fields [71], in order to factorize the sequential likelihood term, incorporate it into the *variational lower bound*, and optimize it via stochastic gradient descent. Furthermore, the work [125] proposes a VAE-based *locally linear* dynamic model, that is inferred from simulated images of a mechanic system with the aim to control said system. Behavior of the latent spaces is modeled by the equation

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}(\bar{\boldsymbol{x}}_t)\boldsymbol{x}_t + \boldsymbol{B}(\bar{\boldsymbol{x}}_t)\boldsymbol{u}_t + \boldsymbol{o}(\bar{\boldsymbol{x}}_t) + \boldsymbol{v}_t. \tag{5.4}$$

The parameters $\boldsymbol{A}$ (state transition), $\boldsymbol{B}$ (control signal transformation) and $\boldsymbol{o}$ (offset) are functions of points $\bar{\boldsymbol{x}}_t$ on the desired state trajectory that are computed from $\bar{\boldsymbol{x}}_t$ using a neural network. This network is trained, along with the rest of the architecture by optimizing an adapted version of the variational lower bound. Similar model assumptions are made in [63], where VAEs are used for state space model identification of dynamic systems. Similarly, the authors of [69] propose an algorithm to employ VAEs as Kalman filters. To do so, they substitute the inference based on the linear parameters of a classical Kalman filter by the variational inference, in which the VAE encoder is used to compute the prospective state distribution.

Subject of an ongoing controversy in building generative models is the question of evaluation. Specifically, it is necessary to assess whether the learned statistics accurately describe the underlying process that has produced the training data. Due to the lack of an objective quantitative measure, it is common to refrain to simple visual inspection of the generated data samples [46]. This approach is therefore also pursued in this chapter. Nevertheless, this chapter aims to provide results that are both transparent and reproducible by evaluating the approach first on sequences for which the behavior is easily predictable, before moving on to more realistic examples that offer more room for disagreement on the synthesis quality.

5.1. Assumptions on the Statistics

The existence of GANs and VAEs has shown that photo-realistic deep generative models for video data can be learned even with limited computational resources and in finite time. This success is not directly applicable when, along with the spatial visual properties, some temporal behavior needs to be learned. One reason for this is the limitation in available data. It is reasonable to assume that we can gather a sufficiently large amount of pictures to train a still-image generation model, but it is difficult to collect a large number of sequences of one particular visual process. Another reason is that observations at one moment can have impact onto the distant future of the visual process behavior. That could potentially imply that the framework needs to process an entire video sequence at once to infer its spatio-temporal statistics, which could quickly become computationally infeasible.

To make the problem of learning the model tractable, this section introduces some assumptions about the visual process at hand. The first assumption will allow us to treat every training sequence as a collection of small sub-sequences, which in turn leads to a decrease in computational demand. Furthermore, it leads to an increase in the amount of training samples, as every sub-sequence can be treated as a training sample. The assumption requires the following two definitions.

Definition 2. *Let the sequence of random variables* $(\mathbf{y}_t)_{t\in\mathbb{Z}}$ *be governed by a stochastic process. It is said that the process has the* Markov property of order T*, if for any* $t \in \mathbb{Z}$ *the conditional distribution of* $\mathbf{y}_t$ *given the* τ *preceding observations is the same for any* $\tau \geq T$*. Formally, given a sequence of observations* $\boldsymbol{y}_1, \dots, \boldsymbol{y}_T$*, and any additional set of observations* $\{\boldsymbol{y}_{t_1}, \boldsymbol{y}_{t_2}, \dots\}$ *with* $T < t_1 < t_2 < \dots$*, the equation*

$$\begin{aligned}&p(\mathbf{y}_t|\mathbf{y}_{t-1} = \boldsymbol{y}_{t-1}, \dots, \mathbf{y}_{t-T} = \boldsymbol{y}_{t-T})\\ =&p(\mathbf{y}_t|\mathbf{y}_{t-1} = \boldsymbol{y}_{t-1}, \dots, \mathbf{y}_{t-T} = \boldsymbol{y}_{t-T}, \mathbf{y}_{t-t_1} = \boldsymbol{y}_{t-t_1}, \mathbf{y}_{t-t_2} = \boldsymbol{y}_{t-t_2}, \dots)\end{aligned} \tag{5.5}$$

holds for any $t \in \mathbb{Z}$*.*

Definition 3. *Let the sequence of random variables* $(\mathbf{y}_t)_{t\in\mathbb{Z}}$ *be governed by a stochastic process. It is said that the process is* stationary *with* order T*, if the joint distribution of any* T *members does not change over time. In more formal terms, consider the index sequence* $t_1, \dots, t_T \in \mathbb{Z}$*. Then, the respective joint probabilities satisfy*

$$p(\mathbf{y}_{t_1}, \dots, \mathbf{y}_{t_T}) = p(\mathbf{y}_{t_1+\tau}, \dots, \mathbf{y}_{t_T+\tau}) \tag{5.6}$$

for any $\tau \in \mathbb{Z}$*.*

Equipped with these two concepts, we can formulate the following assumption.

Assumption 1. *The visual processes of interest are second-order stationary first-order Markov processes.*

The stationarity property accounts for the observation that the spatio-temporal behavior of typical visual processes does not change over time. It is thus a realistic assumption to make. More importantly, the stability property discussed in Section 2.1 that is typically assumed in the context of visual processes is also implied by Assumption 1. The Markov assumption may not be quite accurate in reality, but it greatly facilitates statistical inference. Theoretically, the method presented in this chapter can be generalized to Markov processes of higher orders. This is shown conceptually in Appendix B.2.

The second assumption is related to the technical setup of deep generative models that typically model the latent variables as standard Gaussian noise. To this end, it exploits the ambiguity of the model in Eq. (5.2) with respect to linear transformations of the latent state space. Since Γ_ζ is implemented via a neural network, we can ensure that it accounts for a possible change of basis. The following assumption can thus be made without loss of generality.

Assumption 2. *The latent samples $\boldsymbol{x}_t$ abide a standard Gaussian distribution, i.e., $\mathbf{x}_t \sim \mathcal{N}(\mathbf{x}_t;\, 0, \boldsymbol{I}_n)$.*

If the state transition matrix $\boldsymbol{A}$ is given, and the process is stationary, i.e., $p(\mathbf{x}_t) = p(\mathbf{x}_{t+1})$ for all t, then Assumption 2 essentially identifies the process noise model. Recall that the latent VAR model in Eq. (5.2a) is given by $\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{v}_t$. By taking into account that $\mathbf{v}_t$ is i.i.d. zero-mean standard Gaussian noise, this yields

$$\mathbb{E}_{\mathbf{x}_t}\left[\mathbf{x}_t\mathbf{x}_t^\top\right] = \mathbb{E}_{\mathbf{x}_{t+1}}\left[\mathbf{x}_{t+1}\mathbf{x}_{t+1}^\top\right], \tag{5.7}$$

which is equivalent to

$$\begin{aligned}\boldsymbol{I}_n &= \mathbb{E}_{\mathbf{x}_t,\mathbf{v}_t}\left[(\boldsymbol{A}\mathbf{x}_t + \boldsymbol{B}\mathbf{v}_t)(\boldsymbol{A}\mathbf{x}_t + \boldsymbol{B}\mathbf{v}_t)^\top\right] \\ &= \mathbb{E}_{\mathbf{x}_t}\left[\boldsymbol{A}\mathbf{x}_t\mathbf{x}_t^\top\boldsymbol{A}^\top\right] + \mathbb{E}_{\mathbf{v}_t}\left[\boldsymbol{B}\mathbf{v}_t\mathbf{v}_t^\top\boldsymbol{B}^\top\right] \\ &= \boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{B}^\top.\end{aligned} \tag{5.8}$$

This means that in order to make sure that the latent states $\boldsymbol{x}_t$ remain standard Gaussian in sequential synthesis scenarios, i.e.,

$$\mathbb{E}_{\mathbf{x}_t}\left[\mathbf{x}_t\mathbf{x}_t^\top\right] = \boldsymbol{I}_n,\ \forall t, \tag{5.9}$$

it suffices to ensure the equality $\boldsymbol{I}_n = \boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{B}^\top$. Note that this implies the stability condition

$$\|\boldsymbol{A}\|_2 \leq 1, \tag{5.10}$$

which was discussed in Section 2.1.

5.2. Distributions of Frame Pairs

After introducing Assumption 1 and Assumption 2, let us once again return to the case where the matrix

$$\boldsymbol{Y} = \begin{bmatrix}\boldsymbol{y}_1 & \cdots & \boldsymbol{y}_N\end{bmatrix} \in \mathbb{R}^{d\times N} \tag{5.11}$$

contains an observed sequence of a visual process. It can be seen as an observation of the matrix-valued random variable

$$\mathbf{Y}_N = \begin{bmatrix}\mathbf{y}_1 & \cdots & \mathbf{y}_N\end{bmatrix}. \tag{5.12}$$

Furthermore, let the random variable

$$\mathbf{X}_N = \begin{bmatrix}\mathbf{x}_1 & \cdots & \mathbf{x}_N\end{bmatrix} \tag{5.13}$$

follow the statistics of latent state sequences described by Eq. (5.2a). Consider now a third random variable

$$\tilde{\mathbf{Y}}_N = \begin{bmatrix}\tilde{\mathbf{y}}_1 & \cdots & \tilde{\mathbf{y}}_N\end{bmatrix} := \begin{bmatrix}\Gamma_{\boldsymbol{\zeta}}(\mathbf{x}_1) & \cdots & \Gamma_{\boldsymbol{\zeta}}(\mathbf{x}_N)\end{bmatrix}. \tag{5.14}$$

In order to model the visual process by Eq. (5.2), we need to make sure that the joint probability distributions of $\mathbf{Y}_N$ and $\tilde{\mathbf{Y}}_N$ coincide for any $N \in \mathbb{N}$. Due to Assumption 1, this is equivalent to demanding that the joint probability distributions for two succeeding frames coincide, i.e.,

$$p\left(\mathbf{Y}_2\right) = p\left(\tilde{\mathbf{Y}}_2\right). \tag{5.15}$$

The random variable

$$\tilde{\mathbf{Y}}_2 = \begin{bmatrix}\Gamma_{\boldsymbol{\zeta}}(\mathbf{x}_1) & \Gamma_{\boldsymbol{\zeta}}(\mathbf{x}_2)\end{bmatrix} \tag{5.16}$$

is determined by the according latent variable $\mathbf{X}_2$ for which the joint probability is zero-mean Gaussian. More specifically, we can deduce

$$\mathbb{E}_{\mathbf{x}_t,\mathbf{x}_{t+1}}[\mathbf{x}_t\mathbf{x}_{t+1}^\top] = \mathbb{E}_{\mathbf{x}_t,\mathbf{v}_t}[\mathbf{x}_t(\boldsymbol{A}\mathbf{x}_t + \boldsymbol{B}\mathbf{v}_t)^\top] = \boldsymbol{A}^\top \tag{5.17}$$

due to Assumption 2. For all t, this yields

$$\begin{bmatrix}\mathbf{x}_t \\ \mathbf{x}_{t+1}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_t \\ \mathbf{x}_{t+1}\end{bmatrix}; 0, \begin{bmatrix}\boldsymbol{I}_n & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{I}_n\end{bmatrix}\right). \tag{5.18}$$

If $\boldsymbol{A}$ and $\boldsymbol{\zeta}$ are given, samples from $p(\tilde{\mathbf{Y}}_2)$ can be generated by creating observations of the random variable in Eq. (5.18) and transforming both parts, $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ via $\Gamma_{\boldsymbol{\zeta}}$. To do so, $\boldsymbol{A}$ and $\boldsymbol{\zeta}$ need thus to be learned first. This could be done by means of a VAE. However, VAEs expect standard Gaussian noise at the input and do not provide a natural way to learn a parameterized latent distribution. Therefore, in order to use a VAE to learn these parameters, additional changes to the VAE algorithm are required.

5.3. The Dynamic Layer

The previous section rephrases the problem of learning joint probabilities for sequences of random variables with length $N \in \mathbb{N}$ as learning the probabilities for sequences of length 2. Given $\boldsymbol{Y} \in \mathbb{R}^{d\times N}$, we can collect $N-1$ such sequences by merging each pair of succeeding frames together as

$$\begin{aligned}\boldsymbol{Y}_{\text{pairs}} &= \begin{bmatrix}\boldsymbol{y}_1^2 & \cdots & \boldsymbol{y}_{N-1}^2\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{y}_1 & \cdots & \boldsymbol{y}_{N-1} \\ \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_N\end{bmatrix} \in \mathbb{R}^{2d\times N-1}.\end{aligned} \tag{5.19}$$

Next, we want to use these sequences to learn a CNN $\Gamma_{\boldsymbol{\zeta}}$ and a matrix $\boldsymbol{A}$ such that observations $[\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top]^\top$ of the random variable $[\mathbf{x}_1^\top, \mathbf{x}_2^\top]^\top$ with a distribution as described in Eq. (5.18) are mapped to samples

$$\tilde{\boldsymbol{y}}^2 = \begin{bmatrix} \Gamma_{\boldsymbol{\zeta}}(\boldsymbol{x}_1) \\ \Gamma_{\boldsymbol{\zeta}}(\boldsymbol{x}_2) \end{bmatrix}, \tag{5.20}$$

that abide the same statistics as the columns of $\boldsymbol{Y}_{\text{pairs}}$. Consider now the block matrix

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{I}_n & \mathbf{0} \\ \boldsymbol{A} & \boldsymbol{B} \end{bmatrix}, \tag{5.21}$$

and yet another random variable

$$\mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix} \sim \mathcal{N}(\mathbf{h}; 0, \boldsymbol{I}_{2n}). \tag{5.22}$$

Multiplying $\mathbf{h}$ by $\boldsymbol{F}$ from the left results in a random variable $\boldsymbol{F}\mathbf{h}$ which is zero-mean Gaussian with the covariance matrix

$$\boldsymbol{F}\boldsymbol{F}^\top = \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{B}^\top \end{bmatrix}. \tag{5.23}$$

Now, if $\boldsymbol{A}$ and $\boldsymbol{B}$ fulfill the stationarity condition

$$\boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{B}^\top = \boldsymbol{I}_n, \tag{5.24}$$

then $\boldsymbol{F}\mathbf{h}$ abides the distribution assumption in Eq. (5.18).

By denoting the tuple of parameters that describe Eq. (5.2) by

$$\boldsymbol{\theta} = (\boldsymbol{\zeta}, \boldsymbol{A}, \boldsymbol{B}), \tag{5.25}$$

let us define the function

$$\begin{aligned} f_{\boldsymbol{\theta}} &: \mathbb{R}^{2n} \to \mathbb{R}^{2d}, \\ &\begin{bmatrix} \boldsymbol{h}_1 \\ \boldsymbol{h}_1 \end{bmatrix} \mapsto \begin{bmatrix} \Gamma_{\boldsymbol{\zeta}}(\boldsymbol{h}_1) \\ \Gamma_{\boldsymbol{\zeta}}(\boldsymbol{A}\boldsymbol{h}_1 + \boldsymbol{B}\boldsymbol{h}_2) \end{bmatrix}. \end{aligned} \tag{5.26}$$

If the function $f_{\boldsymbol{\theta}}$ can be learned to map standard Gaussian noise to samples abiding the same distribution as the columns of $\boldsymbol{Y}_{\text{pairs}}$, then Eq. (5.2) with the learned parameters $\boldsymbol{\zeta}, \boldsymbol{A}, \boldsymbol{B}$ accurately describe the visual process at hand, given Assumption 1. The learning can be carried out by employing $f_{\boldsymbol{\theta}}$ as the decoder of a VAE and learning $\boldsymbol{\theta}$ by stochastic gradient descent using the columns of $\boldsymbol{Y}_{\text{pairs}}$ as training data. Figure 5.1 depicts the VAE decoder architecture implementing $f_{\boldsymbol{\theta}}$. The multiplication by $\boldsymbol{F}$ is realized via an upstream dense linear layer. Let us denote this building block the *dynamic layer* and the overall resulting VAE architecture a *Dynamic VAE* (DVAE).

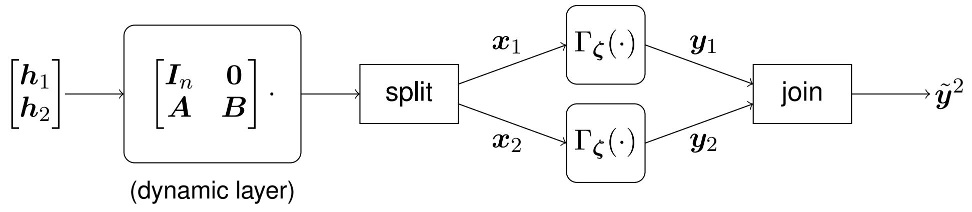


Figure 5.1.: Decoder of a dynamic VAE with a dynamic layer. The input $[\boldsymbol{h}_1^\top, \boldsymbol{h}_2^\top]^\top$ is first linearly transformed by a left-multiplication with the matrix $\boldsymbol{F}$ as defined in Eq. (5.23). This results in a new vector $\boldsymbol{x}$ that is split into two subvectors of the the same size that are both subjected to the neural network implementing the observation function $\Gamma_{\boldsymbol{\zeta}}$. The two outputs are finally concatenated to yield $\tilde{\boldsymbol{y}}^2$. During training, the loss with respect to the input image pair is backpropagated to learn $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{\zeta}$.

5.4. Training the DVAE

As for the procedure and the loss function to be optimized, we need to keep in mind that the stationarity property in Eq. (5.24) needs to remain fulfilled. This can be enforced by means of a simple regularization term added to the variational lower bound. This results in the overall loss

$$\begin{aligned}
&L_{\mathrm{DVAE}}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\zeta}, \boldsymbol{\vartheta}) \\
=&L_{\mathrm{VAE}}((\boldsymbol{\zeta}, \boldsymbol{A}, \boldsymbol{B}), \boldsymbol{\vartheta}) + \lambda \|\boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{B}^\top - \boldsymbol{I}_n\|_F \\
=&\sum_{t=1}^{N-1} \frac{1}{\sigma_{\mathbf{y}}^2} \mathbb{E}_{\mathbf{h}\sim q_{\boldsymbol{\vartheta}}(\mathbf{h}|\mathbf{y}^2=\boldsymbol{y}_t^2)} \left[\|\boldsymbol{y}_t - \Gamma_{\boldsymbol{\zeta}}(\mathbf{h}_1)\|^2 + \|\boldsymbol{y}_{t+1} - \Gamma_{\boldsymbol{\zeta}}(\boldsymbol{A}\mathbf{h}_1 + \boldsymbol{B}\mathbf{h}_2)\|^2\right] \\
&\quad + \mathbf{1}^\top(\boldsymbol{\sigma}_{\mathbf{h}}(\boldsymbol{y}_t^2, \boldsymbol{\vartheta}) - \log \boldsymbol{\sigma}_{\mathbf{h}}(\boldsymbol{y}_t^2, \boldsymbol{\vartheta})) + \|\boldsymbol{\mu}_{\mathbf{h}}(\boldsymbol{y}_t^2, \boldsymbol{\vartheta})\|^2 \\
&+ \lambda \|\boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{B}^\top - \boldsymbol{I}_n\|_F,
\end{aligned} \tag{5.27}$$

which can be optimized via all common stochastic gradient-based algorithms, by randomly sampling from the training pairs $\boldsymbol{y}_1^2, \dots, \boldsymbol{y}_{N-1}^2$ and the posterior model distribution $q_{\boldsymbol{\theta}}(\mathbf{h}|\mathbf{y}^2)$.

5.5. Experiments

The following experiments evaluate the proposed model by generating artificial sequences of previously observed visual processes. Synthesis is performed by sampling from the VAR model described by

$$\begin{aligned}
\boldsymbol{h}_{t+1} &= \boldsymbol{A}\boldsymbol{h}_t + \boldsymbol{B}\boldsymbol{v}_t, \\
\mathbf{v}_t &\sim \text{ i.i.d.}\, \mathcal{N}(\mathbf{v}_t, 0, \boldsymbol{I}_n),
\end{aligned} \tag{5.28}$$

and mapping the latent states to the observation space by means of Γ_{ζ}. The initial latent state $\boldsymbol{h}_0$ is estimated by applying the encoder to a frame pair from the training sequence and obtaining the expected value of the approximate posterior.

Evaluating generative models is always challenging. Goodfellow et al. emphasize this problem in Chapter 20.14 of their introductory book on *Deep Learning* [46]. Most importantly, they note that any generative model can easily mimic a training distribution by simply reproducing the exact same samples that have been used for training. A good evaluation metric would thus need to ensure that it does not reward models which produce data resembling the training samples too much. On the other hand, such a constraint could easily undermine the very purpose of an evaluation metric, since similarity to the training data typically does indicate that the right distribution has been learned.

In the case of video synthesis, the problem is even more difficult, since an evaluation metric would need to reward the visual quality, the temporal consistency and the diversity of the generated video frames, while not favoring models that are too predictable. For instance, to measure the quality of a synthesized frame, one may be tempted to quantitatively evaluate a distance measure, e.g. *peak signal-to-noise-ratio* (PSNR), between a synthesized and a ground truth frame from the training sequence. For two signals $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^d$, the PSNR is defined as the logarithm of the ratio between the maximal possible pixel value and the mean squared error. Assuming normalized images, this amounts to

$$\mathrm{PSNR}(\boldsymbol{y}_1, \boldsymbol{y}_2) = -10 \log_{10} \frac{1}{d} \sum_{i=1}^{d} \left((\boldsymbol{y}_1)_i - (\boldsymbol{y}_2)_i\right)^2 . \tag{5.29}$$

However, such a quality metric would yield the best results for entirely predictable models, which is generally the exact opposite of what video synthesis tries to achieve.

To the author's best knowledge, automated evaluation techniques for video synthesis are still lacking, so research in this field is often performed by visual inspection with the help of human test subjects, as was suggested for instance in [123] and [119]. This was not done in the scope of this work due to lack of resources. Instead, evaluation of the presented approach is focused on visually contrasting the video synthesis results to those produced by baseline methods and to discuss observed shortcomings and advantages. In settings where it is appropriate, these discussions are complemented by some quantitative results. Specifically, the *Fréchet inception distance* (FID) [55] is used to evaluate the visual quality of the synthesis results. The FID score is essentially a distance measure between two Gaussian distributions computed from CNN features of two different sets of images and has become a widely used measure to evaluate the visual quality of GAN generated images.

The remainder of the experimental section is divided into two subsections. The first subsection evaluates the Dynamic VAE on sequences of spatial transformations with little randomness. Artificial sequences are employed for this purpose. In order to create these sequences, simple image data sets are used. The first type of sequences is created using the famous MNIST data set [73] containing 28×28 pixel depictions of handwritten digits. These images are used to create videos of repeating number cycles such as $1234512345\ldots$ The other employed data set is the Small NORB database [74] that contains pictures of miniature figures at different poses and under varying illumination settings. These images are used to create sequences of azimuthal rotations.

The aim of the first subsection is to show that the Dynamic VAE actually learns a proper dynamic model and not just the appearance of individual frames. Primarily, the results are compared to the LDS model in Eq. (1.5) from [38] in order to show the advantage of a non-linear observation space assumption. Besides, a separate model created by learning the VAE before learning the latent VAR parameters by MSE minimization in the latent space is employed, to demonstrate the advantage of learning the observation space and the dynamics jointly. Subsection 5.5 thus also serves as an ablation study that evaluates the advantage of the dynamic layer.

The second subsection evaluates the Dynamic VAE on the problem of dynamic texture synthesis. The results are compared against the LDS model in Eq. (1.5). In order to demonstrate that the model is not only a simple way to describe the stochastics of visual processes, but also competes with state-of-the-art techniques, the results are additionally compared against those produced by the recent *Spatial-Temporal Generative Convnet* (STGCONV) method [127] and the follow-up work on the *Dynamic Generator Model* (DGM) [126], in cases where the baseline results were made available. For the sake of quantitative comparability, the FID score is employed for this series of experiments. Its computation is performed via publicly available third-party code [110].

A Deep Convolutional GAN (DCGAN, [99]) generator has been employed as the foundation for implementing the observation function Γ_{ζ}, used in combination with an affine layer at the input. This layer accounts for changes of bases in the latent space in order to conform with Assumption 2 and reduces the latent dimension to make the search for the transition matrix $\boldsymbol{A}$ feasible. All batch normalization layers have been removed from the network implementing the observation function. The reason for this adaptation is that batch normalization layers remove the mean and normalize the variance of layer outputs, which contradicts the VAE principle that relies on sampling from Gaussian distributions of different first and second-order moments. Specifically, the posterior distribution of the latent variables conditioned over the observations are modeled as Gaussians with different first and second-order statistical parameters. Batch normalization would standardize these parameters, which in turn would make the decoder neglect the observations provided to the encoder during training. As the encoder of the VAE, the discriminator of the DCGAN has been used. Furthermore, the number of encoder output channels have been adapted to be $2n$, where n is the latent dimension of the model.

Experiment	**Resolution**	**Conv. layers**	**Kernel size**	$\sigma_{\mathbf{y}}^2$	λ
MNIST	32×32	4	4	8.0	100.0
Small NORB	96×96	5	6	16.0	100.0
Running Cows	64×64	5	4	10.0	100.0
Salt+Pepper mask	64×64	5	4	4.5	100.0
Rectangular mask	128×128	5	8	8.0	100.0
Dyn. Textures	128×128	5	8	16.0	100.0
Dyn. Textures (Table 5.3)	64×64	5	4	8.0	100.0

Table 5.1.: Experimental configuration

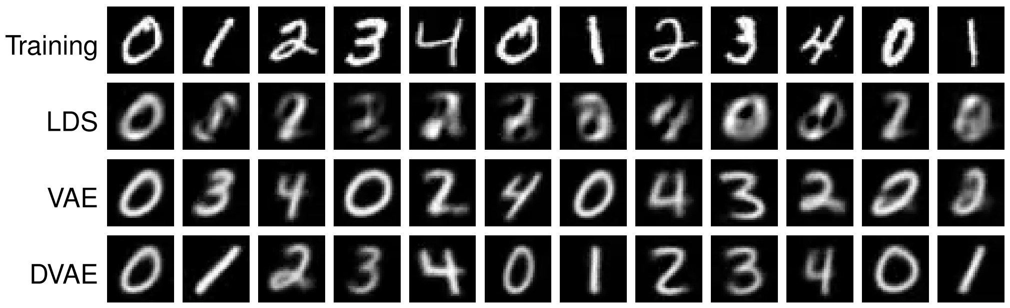


Figure 5.2.: Synthesis of MNIST sequence 0123401234. . . The numbers of the sequence synthesized by the DVAE correspond to the training sequence, while the writing style is random.

The latent dimension was set to $n = 10$ for all experiments. The number of convolutional layers and the size of the convolutional kernel of the decoder output layer (encoder input layer) were varied to match the resolution of the input data. The exact configuration is listed in Table 5.1.

Transformation Sequences

The first experiment investigates DVAE behavior on sequences of handwritten MNIST numbers. Even though this experiment is quite artificial, it provides considerable insight into how well the spatial and temporal statistics of the visual process are captured. Fig. 5.2 depict the synthesis result for the number sequences 0123401234. . . . Additional results can be found in Appendix C.1. The proposed model captures the two essential features of this visual process. On the one hand, the particular number in a frame is entirely deterministic and can be inferred from the previous frame. On the other hand, the way the number is drawn is random and unpredictable. By contrast, the separate VAE+VAR model is only able to capture the

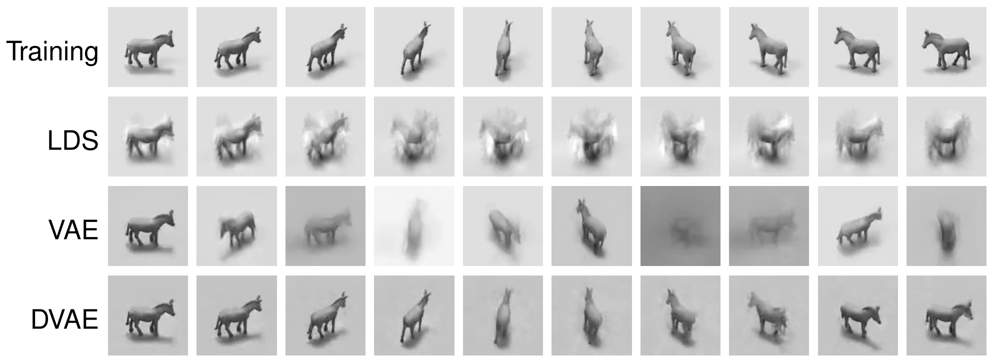


Figure 5.3.: Rotation sequence of Small NORB images (Category 0, Instance 9)

appearance of the numbers and cannot reproduce the number ordering, while the linear model generates hardly recognizable frames.

To investigate the behavior on deterministic sequences of rigid transformations, Category 0 of the *Small NORB* dataset that contains pictures of miniature animals under six different lighting conditions, nine elevational and 18 azimuthal poses is employed. The aim is to synthesize sequences of azimuthal rotations of 20° per frame. Fig. 5.3 depicts the synthesis for instance 9 (*Horse*). More results can be found in Appendix C.1. Overall, the DVAE manages to learn correct frame-to-frame transitions, apart from occasional jumps of 180°. It is interesting that while the separate VAE+VAR model again appears to learn the proper observation space, the linear model does a better job at capturing the rotation of the object, even though it is also learned separately. This contradicts the widely accepted assumption [99, 8] that convolutional deep generative models naturally produce parametrizations that map spatial manipulations of natural images, such as translations and deformations, to linear displacements in the latent space.

Fig. 5.4 depicts synthesis results for the artificially rendered *Running Cows* sequences [127]. The results are compared against STGCONV and classical LDS. It can be observed that although occasional discontinuities occur in the sequence synthesized by the DVAE, as can be seen in the third and forth frame depicted in Fig. 5.4, the overall running movement is accurately reproduced. Unlike LDS or STGCONV, this is done without major visible perturbations. Figure 5.5 visualizes how the different algorithms behave with respect to finer details in the visual process. In the magnified region, it can be observed that LDS tends to blur out fast-moving details, while STGCONV introduces artifacts. Even though it can be also seen that the sequence synthesized by DVAE does not follow the exact pace of the training sequence, the produced results still follow the trained motion pattern without losing too many details or introducing defects.

The DVAE is also capable to learn from incomplete data, i.e., from videos where

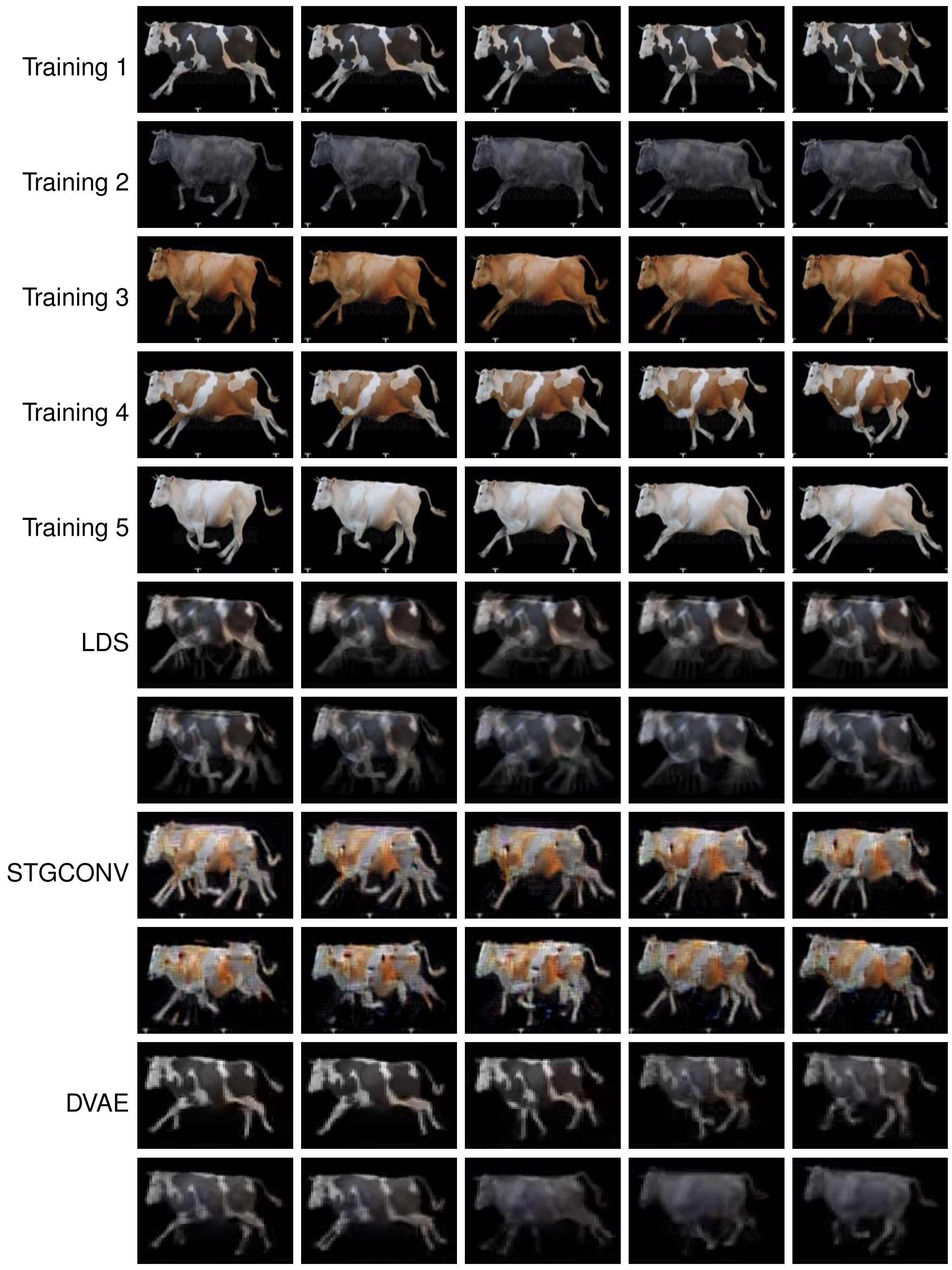


Figure 5.4.: Synthesis results for the *Running Cows* sequence

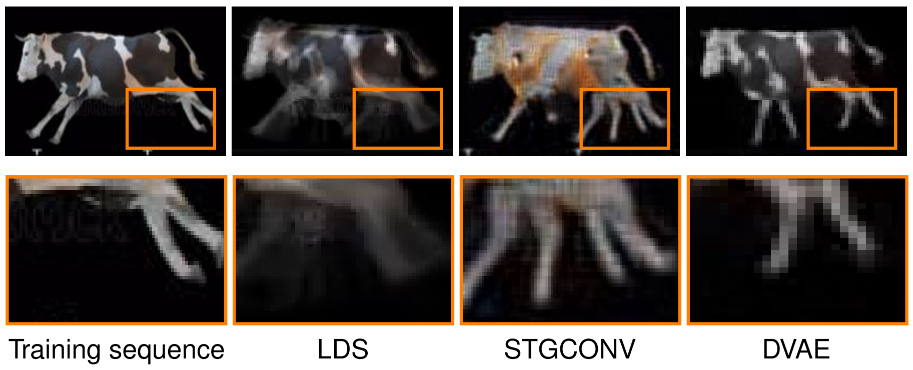


Figure 5.5.: Sixth frame of synthesized videos for *Running Cows* sequence

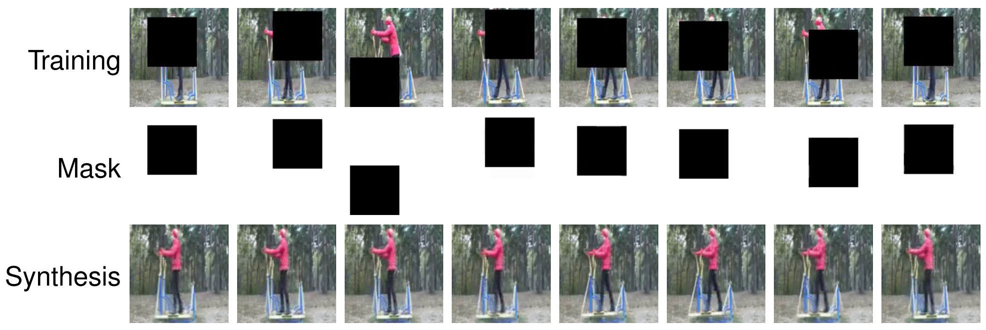


Figure 5.6.: Synthesizing a sequence learned on data obstructed by a rectangular mask

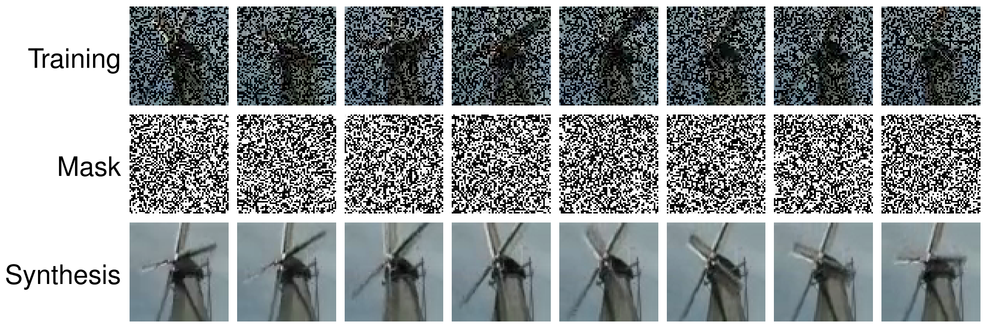


Figure 5.7.: Synthesizing a sequence learned on data obstructed by a salt+pepper mask

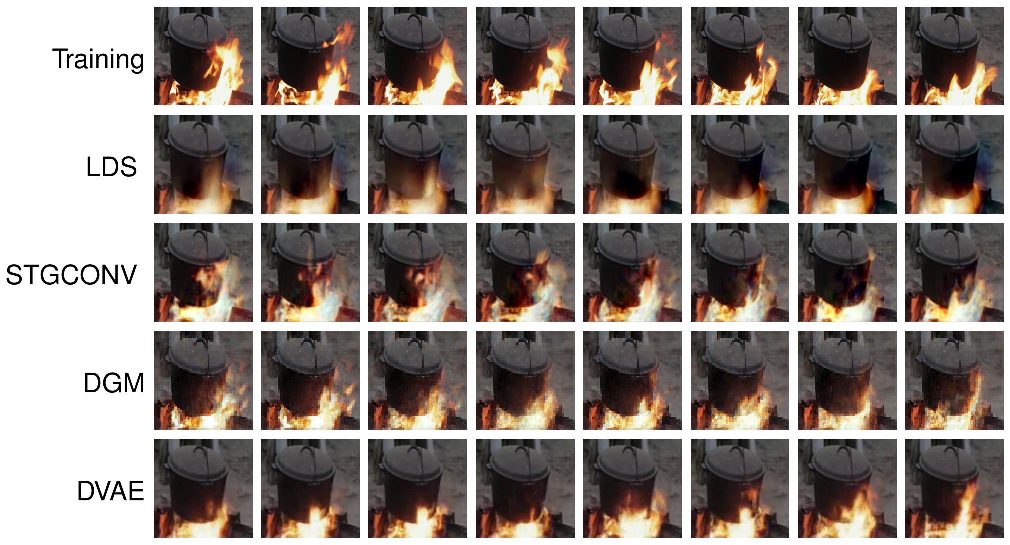


Figure 5.8.: Synthesis of dynamic texture *Fire Pot*

only a subset of pixels are visible while the rest is obstructed. For this purpose, the obstruction mask must be available. In such a scenario, the training objective is only applied to the available pixels. To demonstrate this capability, the DVAE was trained with a partially obstructed sequence and the obstruction mask was provided as an additional input to the DVAE encoder. Fig. 5.6 and Fig. 5.7 depict two synthesis results for this experimental setting. In the former case, a video sequence was obstructed by a square mask that keeps its shape but changes its location over time. The training sequence, the mask and the synthesis result can be observed in Fig. 5.6. Likewise, Fig. 5.7 depicts the synthesis result for 50% salt+pepper obstruction, i.e., masking of pixels that are selected independently with 50% probability in each frame.

Dynamic Textures

Sequences of eleven dynamic textures that were provided in the supplementary material of the texture synthesis work [127] were used for this experiment. The DVAE is capable to capture both the dynamics as well as the appearance of the training videos. Fig. 5.8 depicts synthesis results for an example visual process. Appendix C.1 contains results for ten other dynamic textures. The DVAE results appear slightly blurrier than the sequences produced by STGCONV. However, unlike DVAE, the STGCONV framework has the tendency to reproduce the training sequence frame by frame. Fig. 5.8 illustrates this phenomenon. While, at each time step, the frame synthesized by STGCONV appears to be a perturbed version of the

Sequence	LDS	STGCONV	DGM	DVAE
Flowing Water	233.2	-	-	**163.9**
Boiling Water	**146.6**	-	-	175.8
Sea	113.4	-	-	**64.1**
River	222.8	**103.3**	-	109.8
Mountain Stream	**186.4**	-	-	209.9
Spring Water	329.8	**231.7**	-	235.8
Fountain	243.9	273.9	-	**133.3**
Waterfall	347.0	**236.7**	-	336.1
Washing Machine	**87.8**	-	-	134.8
Flashing Lights	157.8	166.5	265.3	**125.9**
Fire Pot	187.4	188.8	145.4	**115.4**

Table 5.2.: FID scores (compared to training data)

according training frame, DVAE leads to an evolution completely different from the training sequence. DGM does not have this problem, but the frame-to-frame transition can appear slightly unnatural at times, resembling a fading of one frame into the next.

For quantitative comparision, FID scores between synthesized frames and frames from the training sequences have been computed. Two series of experiments have been carried out with this aim. First, 50 frames from each synthesized sequence in the results described above were compared to the frames in the respective training sequence. Table 5.2 summarizes these results. Since the comparison is carried out directly with respect to the frames that were used for training, the results are prone to favor simplistic models that reproduce the ground truth sequence by copying its frames. Despite this flaw of evaluation strategy, the results have been included in this section, as they still provide a visual quality metric and ensure comparability to publicly available results of baseline models.

For the second FID score comparison, the ground truth sequences were split such that the last ten frames of each sequence were not used for training. Rather, the 50 synthesized frames were compared to these remaining ten frames, achieving that the synthesis result is not compared to the training data. Since no baseline data for STGCONV and DGM is available for this configuration, the DVAE is only compared to LDS as synthesis via LDS based models can be easily implemented. Table 5.3 contains the resulting FID scores. To speed up the evaluation, this experiment was carried out using a smaller resolution than in Table 5.2. See Table 5.1 for details.

One should keep in mind that FID, just like any other quantitative measure, provides only limited insight into the quality of a generative model. Beyond that, it considers still-image frame individually rather than in a temporal context. Visual inspection thus remains an important tool for assessing the quality of visual process

Sequence	LDS	DVAE
Flowing Water	227.2	**179.0**
Boiling Water	244.8	**229.2**
Sea	**140.5**	149.1
River	227.8	**160.8**
Mountain Stream	343.1	**305.1**
Spring Water	445.4	**382.3**
Fountain	263.5	**179.4**
Washing Machine	415.8	**412.0**
Flashing Lights	**229.1**	273.8
Fire Pot	**239.9**	253.3

Table 5.3.: FID scores (compared to data not in training sequence)

synthesis. Hence, Appendix C.1 provides an extensive overview of visual synthesis results.

5.6. Discussion

In this chapter, a method to learn generative models of visual processes by adding a dynamic layer to deep generative architectures such as VAEs is presented. Despite having the quite simple mathematical form described by Eq. (5.2), it is capable of reproducing realistic-looking sequences of visual processes.

It is evaluated in experiments on video synthesis and achieves results comparable to state-of-the-art methods for dynamic texture generation. Unlike many common approaches in video synthesis, such as [127] or [119], it does not require optimization over the sequence to be synthesized. Instead, synthesis is performed by sampling from the latent VAR noise, which means that the numerical complexity does not increase supralinearly with the sequence length.

While the chapter has put emphasis on synthesis as an application example, it could also be interesting to investigate other problems in visual processes. Possible applications are for instance the following ones.

- Since the model makes it possible to sample from the distribution of N-length sequences of a visual process, determining the plausibility of an observed sequence for *anomaly detection* could be carried out by evaluating a similarity measure between the observed sequence and a set of Monte Carlo samples created via the model.
- Dynamic texture synthesis has been previously successfully applied in combination with *style transfer* to alter the appearance of generated sequence or

to animate paintings or photographs. Doing so for the present model could be for instance by applying still-image style transfer to synthesized sequences or by altering the observation error model in the variational lower bound from a pixel-based model to one over style-defining moments.

- Since the model is capable of learning from incomplete data, it could be theoretically possible to also apply it to *inverse problems*, such as hole filling or temporal interpolation.

Appendix B discusses potential extensions of the proposed framework.

One shortcoming of the presented model is the low resolution of the synthesized sequences. In principal, increasing the number of layers, or the sizes and strides of the convolutional layers can increase the resulting resolution, but again, this is limited by computational resources. An alternative is to apply *superresolution* methods to the synthesized frames, for instance, approaches that have been successfully applied to still images. The following chapter presents a superresolution method and investigates experimentally, if it can also be applied to synthesized visual process sequences.

6. Post-processing via Super-resolution

One of the most difficult challenges in visual process synthesis is to produce natural-looking sequences of a sufficiently high resolution. With increasing number of pixels, synthesis becomes more prone to failure due to computational demand. The resolution of the sequences produced by the DVAE algorithm in Chapter 5 can be increased by adding more trainable convolutional or fixed upsampling layers. Specifically, increasing the depth of the neural network implementing the observation function Γ_ζ in the previous chapter can increase the number of pixels, since every layer typically performs implicit upsampling of its input. Nevertheless, such adaptations reach their limits sooner or later. This is not just due to hardware constraints such as GPU memory, but also because increasing the number of parameters make models more likely to overfit or get stuck in local minima during optimization.

A potential alternative approach is to increase the resolution by post-processing. In that case, first a low-resolution video sequence would be generated by a video synthesis algorithm. Next, it would be upscaled by a *super-resolution* algorithm. While such an approach also has its limitations, it could theoretically increase the maximal resolution of a video generated by a visual process synthesis method, as the two steps are carried out subsequently.

Super-resolution is a classical inverse problem in computer vision and image processing and can be interpreted as an under-determined system of linear equations [58]. Generally, solving such problems requires additional constraining of the solution set. This is done by incorporating additional model assumptions, so-called *priors*. In their simplest form, priors are handcrafted regularizers that promote certain properties assumed about the data at hand. A prominent example is *total variation* (TV) regularization [103] that favors piecewise smooth solutions. More sophisticated priors require some form of learning procedure applied to the input image in order to infer additional structural information. For instance, it is common to assume that localized patches of a natural image have a sparse representation with respect to some dictionary [129].

Nowadays, deep learning based approaches are considered the state of the art at solving inverse computer vision problems, including super-resolution [72, 51, 75]. However, unlike the traditional "shallow" algorithms mentioned above, they rely on external image data sets to be trained on.

In any case, super-resolution increases the resolution of a given image by adding pixels to it based on certain assumptions about the properties these pixels should fulfill. These assumptions can be explicit, as in the case of TV that is designed following the observation that natural images rarely exhibit any abrupt changes in color. They can also be gathered from example data, as is the case for many deep learning based approaches. Since these assumptions are not necessarily considered in the design of the visual process model, they can be exploited to further enhance its visual quality.

While methods for the solution of inverse problems in deep learning often involve large training sets, it was mentioned before that training is only one of several factors that explain the success of deep learning. Notably, the *deep image prior* (DIP) [121] has recently demonstrated that a deep CNN can still provide superior performance on different kinds of inverse problems, even though it is trained exclusively on the input image itself, thus leveraging the advantages of traditional and deep learning algorithms.

The aim of this chapter is to develop a super-resolution technique and to investigate to what extent it is applicable to improve the visual quality of video synthesis results. This super-resolution algorithm is based on an alternative training objective of DIP. When evaluated on still images, it outperforms the classical DIP formulation in terms of the PSNR score on common super-resolution benchmarks. It also yields superior results when compared against other super-resolution approaches that do not rely on additional data. Solely deep learning methods that have been trained on large image datasets are shown to significantly outperform the approach described in this chapter.

In the theoretical descriptions of this chapter, the proposed algorithm is not discussed with regard to visual processes, since it does not conceptually differ from any common still-image super-resolution technique. The experimental section then provides an assessment of applicability to visual processes. Since the targeted application scenario revolves around video synthesis, objective assessment is challenging due to the lack of ground truth data. As in Chapter 5, the evaluation thus refrains to a combination of FID score comparison and visual inspection.

6.1. Super-resolution as an Inverse Problem

Super-resolution describes the process of reconstructing a high-resolution signal $\boldsymbol{z} \in \mathbb{R}^{\tau d}$ from a low-resolution version $\boldsymbol{y} \in \mathbb{R}^{d}$ with $\tau \in \mathbb{N}$ being the sampling rate. For the sake of simplicity, let us treat $\boldsymbol{y}$ and $\boldsymbol{z}$ as one-dimensional signals, since generalization to 2D images is straightforward. It is assumed that the signal $\boldsymbol{y}$ results from filtering of $\boldsymbol{z} \in \mathbb{R}^{\tau d}$ with a filter $\boldsymbol{h} \in \mathbb{R}^{k\tau}$ with $k \leq d$, followed by a downsampling operation with the factor τ. To formalize this relationship, the downsampling operation is defined as

$$\begin{gathered}\mathrm{DS}_\tau \colon \mathbb{R}^{\tau d} \to \mathbb{R}^{d}, \\ \begin{bmatrix} y_1 & y_2 & \cdots & y_{d\tau-1} & y_{d\tau}\end{bmatrix}^\top \mapsto \begin{bmatrix} y_\tau & y_{2\tau} & \cdots & y_{(d-1)\tau} & y_{d\tau}\end{bmatrix}^\top .\end{gathered} \tag{6.1}$$

Additionally, let the star operator $\star$ denote discrete cross-correlation. The two signals $\boldsymbol{z}$ and $\boldsymbol{y}$ are then related via the equation

$$\boldsymbol{y} = \mathrm{DS}_\tau[\boldsymbol{h} \star \boldsymbol{z}]. \tag{6.2}$$

Eq. (6.2) is a linear operation and can thus be written as a matrix-vector multiplication. If we split up the filter $\boldsymbol{h}$ into subfilters $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_k \in \mathbb{R}^\tau$ of the same size, i.e., if we write it as

$$\boldsymbol{h}^\top = \begin{bmatrix} \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top \end{bmatrix}, \tag{6.3}$$

then we can define the matrix

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top & & & \\ & \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top & & \\ & & \ddots & \ddots & & \ddots & \ddots & \\ & & & \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top \\ & & & & \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top \\ & & & & & \ddots & \ddots & \vdots \\ & & & & & & \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top \\ & & & & & & & \boldsymbol{h}_1^\top \end{bmatrix}. \tag{6.4}$$

Note that $\boldsymbol{H} \in \mathbb{R}^{d \times \tau d}$ can be generated by first creating a Toeplitz matrix in $\mathbb{R}^{\tau d \times \tau d}$ from the the row vector $\boldsymbol{h}$ and sampling every τth row from it. Using this definition, Eq. (6.2) can be rewritten as the matrix-vector product

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{z}. \tag{6.5}$$

Clearly, Eq. (6.5) is an under-determined system of equations. Uniquely recovering $\boldsymbol{z}$ exclusively from the observation $\boldsymbol{y}$ is thus generally not possible, unless we can make additional assumptions about $\boldsymbol{z}$. Like many other inverse problems, super-resolution can be phrased as the problem of choosing, formalizing and implementing appropriate assumptions with regards to $\boldsymbol{z}$.

6.2. The Deep Image Prior

As the proposed super-resolution algorithm is based on DIP, this section reviews how DIP is used to solve inverse problems.

Linear inverse problems are typically characterized by a (full-rank) *degradation* operator $\boldsymbol{A}_{\mathrm{IP}} \in \mathbb{R}^{d \times n_{\mathrm{IP}}}, d < n_{\mathrm{IP}}$ and an optional noise term $\boldsymbol{\eta} \in \mathbb{R}^d$. The observation $\boldsymbol{y}$ is thus related to the original signal $\boldsymbol{z}$ via

$$\boldsymbol{y} = \boldsymbol{A}_{\mathrm{IP}}\boldsymbol{z} + \boldsymbol{\eta}. \tag{6.6}$$

DIP is a framework for solving inverse problems described by Eq. (6.6). To this end, it employs a CNN which generates the reconstruction from a fixed noise vector $\boldsymbol{r} \in \mathbb{R}^{\tilde{d}}$. Let us write the function described by the CNN as

$$T_{\boldsymbol{\nu}} : \mathbb{R}^{\tilde{d}} \to \mathbb{R}^{d}, \tag{6.7}$$

where $\boldsymbol{\nu}$ denotes its trainable parameters and the set containing all realizable functions using the same architecture as

$$\mathcal{T} = \left\{ T_{\boldsymbol{\nu}} \colon \mathbb{R}^{\tilde{d}} \to \mathbb{R}^{d} \mid \boldsymbol{\nu} \in \mathbb{R}^{D} \right\}. \tag{6.8}$$

Given some randomly chosen input $\boldsymbol{r}$, DIP optimizes the objective function

$$L_{\mathrm{DIP}}(\boldsymbol{\nu}) = \|\boldsymbol{A}_{\mathrm{IP}} T_{\boldsymbol{\nu}}(\boldsymbol{r}) - \boldsymbol{y}\|^2, \tag{6.9}$$

and returns

$$\hat{\boldsymbol{z}}_{\mathrm{DIP}} = T_{\hat{\boldsymbol{\nu}}}(\boldsymbol{r}) \tag{6.10}$$

with

$$\hat{\boldsymbol{\nu}} = \underset{\boldsymbol{\nu} \in \mathbb{R}^{D}}{\arg\min}\, L_{\mathrm{DIP}}(\boldsymbol{\nu}) \tag{6.11}$$

as the approximated solution of Eq. (6.6). That is to say, the inverse problem is solved by choosing $\boldsymbol{z} = T_{\boldsymbol{\nu}}(\boldsymbol{r})$ such that $T_{\boldsymbol{\nu}}$ is the element of $\mathcal{T}$ that minimizes the squared error in Eq. (6.9). Surprisingly, this approach produces impressively good results for a large set of inverse problems, including super-resolution [121].

6.3. Proposed Method

To understand the proposed algorithm, it can be helpful to interpret Eq. (6.9) in terms of the involved sets. If the problem is noiseless, such as is often the case for super-resolution, then the solution set of Eq. (6.6) is an affine space of the form

$$\mathbb{V} = \{\boldsymbol{A}_{\mathrm{IP}}^{+}\boldsymbol{y} + \tilde{\boldsymbol{z}} \mid \tilde{\boldsymbol{z}} \in \mathbb{R}^{n_{\mathrm{IP}}} \text{ s.t. } \boldsymbol{A}_{\mathrm{IP}}\tilde{\boldsymbol{z}} = 0\}. \tag{6.12}$$

Conversely, the *DIP set*, i.e., the set to which a solution computed by DIP given an input noise vector $\boldsymbol{r}$ is constrained, can be written as

$$\mathbb{T}_{\boldsymbol{r}} = \{T_{\boldsymbol{\nu}}(\boldsymbol{r}) \mid T_{\boldsymbol{\nu}} \in \mathcal{T}\}. \tag{6.13}$$

Given a matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n_{\mathrm{IP}}}$, let us now define the $\boldsymbol{W}$-*weighted* distance $d_{\boldsymbol{W}}$ of a point $\boldsymbol{z} \in \mathbb{R}^{n_{\mathrm{IP}}}$ to a set $\mathcal{M} \subset \mathbb{R}^{n_{\mathrm{IP}}}$ as

$$d_{\boldsymbol{W}}(\boldsymbol{z}, \mathcal{M}) = \min_{\tilde{\boldsymbol{z}} \in \mathcal{M}} \|\boldsymbol{W}(\boldsymbol{z} - \tilde{\boldsymbol{z}})\|^2. \tag{6.14}$$

The solution computed by DIP is thus essentially the point $\hat{\boldsymbol{z}}_{\mathrm{DIP}} \in \mathbb{T}_{\boldsymbol{r}}$ that minimizes the $\boldsymbol{A}_{\mathrm{IP}}$-weighted distance to $\mathbb{V}$. It can be expressed as

$$\hat{\boldsymbol{z}}_{\mathrm{DIP}} = \underset{\boldsymbol{z} \in \mathbb{T}_{\boldsymbol{r}}}{\arg\min}\, d_{\boldsymbol{A}_{\mathrm{IP}}}(\boldsymbol{z}, \mathbb{V}). \tag{6.15}$$

Unfortunately, constraining the solution to lie on $\mathbb{T}_{\boldsymbol{r}}$ has some drawbacks. Note that $\mathbb{V}$ and $\mathbb{T}_{\boldsymbol{r}}$ are two sets that are typically disjoint in practice. This means that a solution $\hat{\boldsymbol{z}}_{\mathrm{DIP}}$ recovered via DIP from an observation $\boldsymbol{y}$ would not fulfill the property

$$\boldsymbol{A}_{\mathrm{IP}} \hat{\boldsymbol{z}}_{\mathrm{DIP}} = \boldsymbol{y}. \tag{6.16}$$

This may be desirable, when the noise perturbation by $\boldsymbol{\eta}$ is significant, but does not make much sense in inverse problems where $\boldsymbol{\eta} \approx 0$, such as in noiseless super-resolution. This problem could be overcome by projecting $\hat{\boldsymbol{z}}_{\mathrm{DIP}}$ onto $\mathbb{V}$, once it is determined. However, there is another problem with formulation Eq. (6.15) that occurs when the objective is employed in combination with a regularizer $R\colon \mathbb{R}^{n_{\mathrm{IP}}} \to \mathbb{R}$. Such a term can be added to the loss function in order to enforce solutions that fulfill certain complementary properties. Adjusting Eq. (6.15) (and thus Eq. (6.9)) in such a way yields

$$\hat{\boldsymbol{z}}_{\mathrm{DIP,reg}} = \underset{\boldsymbol{z} \in \mathbb{T}_{\boldsymbol{r}}}{\arg\min}\, d_{\boldsymbol{A}_{\mathrm{IP}}}(\boldsymbol{z}, \mathbb{V}) + \lambda_{\mathrm{reg}} R(\boldsymbol{z}), \tag{6.17}$$

where $\lambda_{\mathrm{reg}} > 0$ is a tuning parameter. Consider now the extreme case where

$$R(\boldsymbol{z}) = \mathrm{const.} \qquad \forall \boldsymbol{z} \in \mathbb{T}_{\boldsymbol{r}} \tag{6.18}$$

holds. Then, adding R to the loss has no impact on the outcome no matter the weight λ_{reg} assigned to it. Practically, this problem occurs with regularizers that enforce certain properties on the high frequencies contained the image. Since DIP tends to return smooth images, this additional guidance gets lost.

An alternative formulation to Eq. (6.9) is

$$L_{\mathrm{btwn}}(\boldsymbol{\nu}, \boldsymbol{z}) = \|T_{\boldsymbol{\nu}}(\boldsymbol{r}) - \boldsymbol{z}\|^2 + \lambda_{\mathrm{reg}} \|\boldsymbol{A}_{\mathrm{IP}} \boldsymbol{z} - \boldsymbol{y}\|^2, \tag{6.19}$$

where $\lambda_{\mathrm{reg}} > 0$ is a tuning parameter. Eq. (6.19) searches for a solution somewhere between $\mathbb{V}$ and $\mathbb{T}_{\boldsymbol{r}}$. However, this formulation does not account for the important case $\boldsymbol{\eta} = 0$.

Instead, let us consider an inversion of the perspective in Eq. (6.15), where instead of searching for a solution on $\mathbb{T}_{\boldsymbol{r}}$ that minimizes a distance to $\mathbb{V}$, we search for a solution on $\mathbb{V}$ that minimizes a distance to $\mathbb{T}_{\boldsymbol{r}}$. Given some weight matrix $\boldsymbol{B}_{\mathrm{IP}} \in \mathbb{R}^{m \times n_{\mathrm{IP}}}, m \leq n_{\mathrm{IP}}$, we can formulate the optimization problem as

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z} \in \mathbb{V}}{\arg\min}\, d_{\boldsymbol{B}_{\mathrm{IP}}}(\boldsymbol{z}, \mathbb{T}_{\boldsymbol{r}}). \tag{6.20}$$

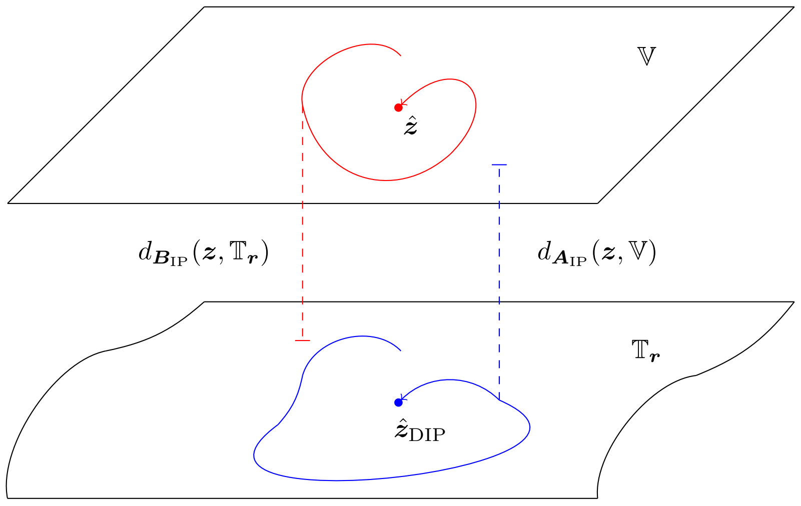

$\mathbb{V}$
$\hat{\boldsymbol{z}}$
$d_{\boldsymbol{B}_{\mathrm{IP}}}(\boldsymbol{z}, \mathbb{T}_{\boldsymbol{r}})$
$d_{\boldsymbol{A}_{\mathrm{IP}}}(\boldsymbol{z}, \mathbb{V})$
$\mathbb{T}_{\boldsymbol{r}}$
$\hat{\boldsymbol{z}}_{\mathrm{DIP}}$

Figure 6.1.: Optimization procedures of Eq. (6.15) and Eq. (6.20). While classical DIP (blue) searches on the CNN set $\mathbb{T}_{\boldsymbol{r}}$ for a point that minimizes the distance to $\mathbb{V}$ (blue dashed line), the proposed formulation (red) searches for a point on $\mathbb{V}$ that minimizes the distance to $\mathbb{T}_{\boldsymbol{r}}$ (red dashed line).

Now, the solution is guaranteed to fulfill exactly the conditions posed by the inverse problem in Eq. (6.6). Additionally we can add a regularizer $R(\boldsymbol{z})$ to Eq. (6.20) via

$$\hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}\in\mathbb{V}} d_{\boldsymbol{B}_{\mathrm{IP}}}(\boldsymbol{z}, \mathbb{T}_{\boldsymbol{r}}) + \lambda_{\mathrm{reg}} R(\boldsymbol{z}). \tag{6.21}$$

The case

$$R(\boldsymbol{z}) = \mathrm{const.} \qquad \forall \boldsymbol{z} \in \mathbb{V}, \tag{6.22}$$

would once again render the regularizer ineffective. However, in that case, R does not provide any additional knowledge about the inverse problem anyway, since any possible property enforced by R is already described by the inverse problem itself. One further advantage is that $T_{\boldsymbol{\nu}}(\boldsymbol{r})$ is not regularized directly. That means the optimization problem can be written in such a way that the gradient of R does not need to be backpropagated through $\boldsymbol{\nu}$. This can avoid numerical complications that tend to happen in neural network optimization when computing the gradient of exotic loss functions for parameters of deep neural nets. Fig. 6.1 visualizes the difference between Eq. (6.15) and Eq. (6.20).

6.4. Implementation with CNNs

Returning to the super-resolution problem in Eq. (6.5), let $\boldsymbol{G} \in \mathbb{R}^{(\tau-1)d\times\tau d}$ be a full-rank matrix with columns that span the kernel $\ker(\boldsymbol{H})$ of $\boldsymbol{H}$. Then, we can rewrite the solution set $\mathbb{V}$ as

$$\mathbb{V} = \{\boldsymbol{H}^{+}\boldsymbol{y} + \boldsymbol{G}^{\top}\boldsymbol{s} | \boldsymbol{s} \in \mathbb{R}^{(\tau-1)d}\}. \tag{6.23}$$

Rewriting the super-resolution problem following the paradigm in Eq. (6.20) while constraining $\boldsymbol{z}$ to $\mathbb{V}$, yields the weighted set distance

$$d_{\boldsymbol{B}_{\mathrm{IP}}}(\boldsymbol{H}^{+}\boldsymbol{y} + \boldsymbol{G}^{\top}\boldsymbol{s}, \mathbb{T}_{\boldsymbol{r}}) = \min_{\boldsymbol{\nu}\in\mathbb{R}^{D}} \|\boldsymbol{B}_{\mathrm{IP}}(\boldsymbol{H}^{+}\boldsymbol{y} + \boldsymbol{G}^{\top}\boldsymbol{s} - T_{\boldsymbol{\nu}}(\boldsymbol{r}))\|^2. \tag{6.24}$$

We can thus define a new loss function by

$$L_{\mathrm{DIP2}}(\boldsymbol{s}, \boldsymbol{\nu}) = \|\boldsymbol{B}_{\mathrm{IP}}(\boldsymbol{H}^{+}\boldsymbol{y} + \boldsymbol{G}^{\top}\boldsymbol{s} - T_{\boldsymbol{\nu}}(\boldsymbol{r}))\|^2. \tag{6.25}$$

Theoretically, the matrix $\boldsymbol{B}_{\mathrm{IP}}$ can be simply chosen to be the identity matrix. However, this may not be a working choice in practice. To illustrate the problem of such a choice, let us consider the case where the filter $\boldsymbol{h}$ is simply a dirac-delta impulse, i.e., a standard basis vector. Then, $\boldsymbol{H}$ consists of every τ-th row of the identity matrix $\boldsymbol{I}_{\tau d}$ and $\boldsymbol{G}$ contains all the remaining rows that are not contained in $\boldsymbol{H}$. In that case, both $\boldsymbol{y}$ and $\boldsymbol{s}$ correspond to pixel values of the image to be reconstructed. Practically, the vector $\boldsymbol{s}$ needs to be initialized with some meaningless values. These values may have significant impact on the overall loss so that $\boldsymbol{\nu}$ is adapted in a way such that $T_{\boldsymbol{\nu}}(\boldsymbol{r})$ approximates $\boldsymbol{s}$ for the respective values. This leads to a situation, in which $\boldsymbol{s}$ does no longer contribute sufficiently to the loss function and thus remains close to its initialization. As a consequence, the computed solution is filled in using meaningless initialization pixels. This problem can be mitigated by using a weight matrix $\boldsymbol{B}_{\mathrm{IP}}$ that makes sure that $\boldsymbol{s}$ does not have a significant impact on the loss during early phases of the optimization, so that the optimization is not heavily affected by the error with respect to $\boldsymbol{s}$ during early iterations.

The matrices $\boldsymbol{B}_{\mathrm{IP}}, \boldsymbol{G}, \boldsymbol{H}$ are typically very large for real-world images. Backpropagating through such large matrices is computationally not feasible, unless they can be implemented as (transpose-)convolutional layers or similar sparse structures. This is definitely possible for $\boldsymbol{H}$, since it essentially describes a filtering operation. Luckily, $\boldsymbol{G}$ can be also constructed in an according manner. Note that $\boldsymbol{G} \in \mathbb{R}^{(\tau-1)d\times\tau d}$ can be any full-rank matrix that fulfills the condition

$$\boldsymbol{H}\boldsymbol{G}^{\top} = 0. \tag{6.26}$$

Now, consider a set of filters $\boldsymbol{g}_1, \dots, \boldsymbol{g}_{\tau-1} \in \mathbb{R}^{k\tau}$ that fulfill the properties

$$\begin{bmatrix} \boldsymbol{h}_k^\top & & & & \\ \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top & & & \\ \vdots & \ddots & \ddots & & \\ \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top & \\ \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top & \boldsymbol{h}_k^\top \\ & \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top & \cdots & \boldsymbol{h}_{k-1}^\top \\ & & \ddots & \ddots & \vdots \\ & & & \boldsymbol{h}_1^\top & \boldsymbol{h}_2^\top \\ & & & & \boldsymbol{h}_1^\top \end{bmatrix} \boldsymbol{g}_i = 0, \qquad \forall i \tag{6.27}$$

and

$$\boldsymbol{g}_i^\top \boldsymbol{g}_j = 0,\ \boldsymbol{g}_i^\top \boldsymbol{g}_i = 1, \qquad \forall i, j,\ i \neq j. \tag{6.28}$$

Such a set can be created, for instance, using the SVD of the matrix in Eq. (6.27).

From $\boldsymbol{g}_1, \dots, \boldsymbol{g}_{\tau-1}$, a set of matrices $\boldsymbol{G}_1, \dots, \boldsymbol{G}_{\tau-1}$ can be generated, according to the same principle as in Eq. (6.4). Then, the block matrix

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_1 & \cdots & \boldsymbol{G}_{\tau-1} \end{bmatrix} \in \mathbb{R}^{(\tau-1)d \times \tau d} \tag{6.29}$$

is full-rank and fulfills the property in Eq. (6.26). Thus, multiplications involving $\boldsymbol{G}$ can be implemented using convolutional or transpose-convolutional layers, because the block components of $\boldsymbol{G}$ essentially realize the cross-correlation with the filters $\boldsymbol{g}_1, \dots, \boldsymbol{g}_{\tau-1}$.

Finally, let us fix

$$\boldsymbol{B}_{\mathrm{IP}} = \begin{bmatrix} \boldsymbol{H}^\top & \sqrt{\lambda_{\boldsymbol{G}}} \boldsymbol{G}^\top \end{bmatrix}^\top, \tag{6.30}$$

where $\lambda_{\boldsymbol{G}} > 0$ is a parameter that controls the impact of obscured or perturbed pixels that need to be interpolated by $T_{\boldsymbol{\nu}}$.

That way, we can rewrite the objective in Eq. (6.25) as

$$L_{\mathrm{DIP2}}(\boldsymbol{s}, \boldsymbol{\nu}) = \|\boldsymbol{H}(\boldsymbol{H}^+ \boldsymbol{y} - T_{\boldsymbol{\nu}}(\boldsymbol{r}))\|^2 + \lambda_{\boldsymbol{G}} \|\boldsymbol{G}(\boldsymbol{G}^\top \boldsymbol{s} - T_{\boldsymbol{\nu}}(\boldsymbol{r}))\|^2. \tag{6.31}$$

Empirical observations indicate that the tuning parameter $\lambda_{\boldsymbol{G}}$ should be chosen one or a few orders of magnitude smaller than 1. As outlined in the previous section, the optimization then does not get stuck at early estimates of $\boldsymbol{s}$, thus avoiding early and meaningless local minima.

Minimizing Eq. (6.31) yields the optimum $\hat{\boldsymbol{\nu}}_{\mathrm{DIP2}}, \hat{\boldsymbol{s}}_{\mathrm{DIP2}}$. The superresolved image is then obtained via

$$\hat{\boldsymbol{z}}_{\mathrm{DIP2}} = \boldsymbol{H}^+ \boldsymbol{y} + \boldsymbol{G}^\top \hat{\boldsymbol{s}}_{\mathrm{DIP2}}. \tag{6.32}$$

6.5. Experiments for Super-resolution with Ground Truth

Before examining the proposed method on synthesized visual processes, this section establishes that the described algorithm is an effective method when it comes to classical super-resolution. For this purpose, the method is compared to other common super-resolution techniques on widespread still-image datasets and dynamic texture sequences.

Still-image Super-resolution

The proposed algorithm was evaluated on the widely employed super-resolution benchmarks Set5 [17] and Set14 [131] using magnification factor $\tau = 4$ and $\lambda_{\boldsymbol{G}} = 0.1$. Code from the official project website [120] has been reused for the DIP implementation to make sure any improvement over classical DIP is not due to a different choice of hyperparameters. A Lanczos filter was used as $\boldsymbol{h}$, even though the actual filter that was used to perform the downsampling is unknown. Since many super-resolution algorithms operate on gray-scale images only, it is common to calculate the PSNR exclusively on the luminance channel of an image. Here, the PSNR is computed on all three RGB channels. This explains why values for the DIP result reported in [121] are slightly higher.

Both DIP and the proposed method are evaluated once without any regularization, as well as using a TV prior as a regularizer. For a 2D image $\boldsymbol{z}$ with row and column indexes i, j, this regularizer is defined as

$$R_{\mathrm{TV}}(\boldsymbol{z}) = \sum_{i,j} \sqrt{(\boldsymbol{z}(i,j) - \boldsymbol{z}(i,j-1))^2 + (\boldsymbol{z}(i,j) - \boldsymbol{z}(i-1,j))^2}. \tag{6.33}$$

For the classical DIP formulation and using a weight parameter λ_{TV}, the loss results in

$$L_{\mathrm{DIP,TV}}(\boldsymbol{\nu}) = L_{\mathrm{DIP}}(\boldsymbol{\nu}) + \lambda_{\mathrm{TV}} R_{\mathrm{TV}}(T_{\boldsymbol{\nu}}(\boldsymbol{r})). \tag{6.34}$$

The loss of the proposed method amounts to

$$L_{\mathrm{DIP2,TV}}(\boldsymbol{s}, \boldsymbol{\nu}) = L_{\mathrm{DIP2}}(\boldsymbol{s}, \boldsymbol{\nu}) + \lambda_{\mathrm{TV}} R_{\mathrm{TV}}(\boldsymbol{s}). \tag{6.35}$$

In the following experiments, the value $\lambda_{\mathrm{TV}} = 10^{-3}/(\tau d)$ was used. Note how in Eq. (6.35), the regularizer is not a function of $\boldsymbol{\nu}$, as opposed to Eq. (6.34). This means that the gradient of the numerically intricate square root does not need to be backpropagated through the neural network weights, which reduces the chance of numerical failure.

Table 6.1 presents the PSNR results of the proposed method in comparison to the learning-free DIP and bicubic interpolation, as well as the deep learning based LapSRN [72]. The proposed approach almost consistently leads to an improvement in performance over classical DIP. Additionally, using a TV regularizer is less

		Bicubic	DIP	DIP+TV	Proposed	Pro-posed+TV	Lap-SRN
Set 5	Baby	30.43	29.48	nan	31.29	*31.35*	**32.02**
	Bird	28.09	28.91	29.11	29.68	*29.83*	**30.42**
	Butterfly	20.90	24.46	24.65	24.71	*24.83*	**25.52**
	Head	28.72	28.23	28.53	28.62	*28.78*	**29.62**
	Woman	25.39	26.72	26.85	27.23	*27.25*	**29.24**
	Avg.	26.71	27.56	-	28.30	*28.41*	**29.36**
Set 14	Baboon	20.26	20.16	20.19	*20.47*	20.46	**20.60**
	Barbara	23.53	23.70	23.67	*24.02*	23.98	**24.11**
	Bridge	23.07	23.01	23.01	23.39	*23.46*	**23.93**
	Coastguard	24.00	24.19	24.31	*24.34*	24.32	**24.81**
	Comic	20.16	20.80	20.88	20.93	*20.94*	**21.35**
	Face	*28.72*	28.26	28.46	28.68	*28.72*	**29.62**
	Flowers	23.64	24.20	24.17	24.73	*24.80*	**25.23**
	Foreman	26.02	27.25	*27.56*	27.50	27.04	**28.98**
	Lenna	28.36	28.53	nan	29.46	*29.48*	**29.88**
	Man	24.42	24.61	24.66	25.19	25.25	**25.99**
	Monarch	26.25	28.09	nan	29.22	*29.30*	**30.13**
	Pepper	27.24	27.53	27.61	*28.29*	28.26	**28.79**
	Ppt3	20.31	22.51	nan	*22.80*	nan	**23.58**
	Zebra	22.79	24.17	24.02	*24.87*	24.60	**25.65**
	Avg.	24.20	24.79	-	*25.28*	-	**25.90**

Table 6.1.: PSNR values for 4x super-resolution. Best and second-best results are written in **bold**, and *italic*, respectively.

prone to numerical failures when employed with the proposed method as opposed to the original DIP formulation. Figure 6.2 illustrates why the proposed formulation tends to outperform classical DIP. The set $\mathbb{T}_r$ tends to not capture certain directional structures at very high frequencies, which can be also empirically observed in experimental results presented in earlier works, e.g. [121], and is exploited in denoising problems. Restricting the solution to $\mathbb{T}_r$ thus causes a blurring out of such structures. This is why, for instance, the DIP struggles to reproduce fine details such as eyelashes, while the proposed method does not have this problem to that extent.

Application to Visual Processes

The preceding discussion indicates that the proposed method outperforms standard learning-free interpolation methods such as bicubic interpolation but also the basic DIP algorithm as described in [121] when evaluated on still-image datasets. A similar observation can be made with regards to video sequences.

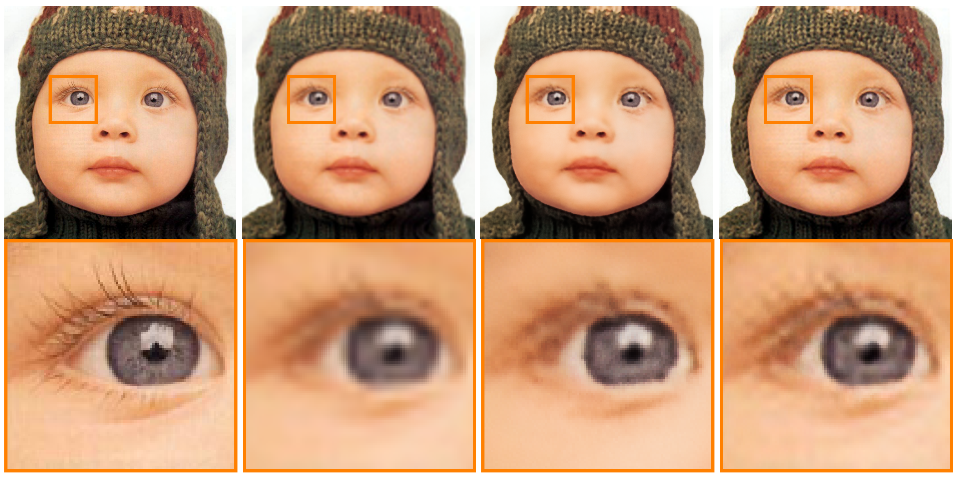


Figure 6.2.: Reconstruction results for “Baby”. The proposed method manages to reconstruct high-frequency details such as eyelashes that tend to be blurred out by the classical DIP framework.

To demonstrate this, a super-resolution experiment has been carried out using the dynamic texture sequences from [127] that were already investigated in Chapter 5. The video sequences were first downsized to the resolution 256×256. Then, the first nine frames were extracted to create ground-truth subsequences. These sequences were then further downscaled with the magnification factor $\tau = 2$ to create sequences with resolution 128×128 which were then fed to the respective super-resolution algorithm. The classical DIP and the proposed method were employed assuming a box filter for $\boldsymbol{h}$, and the weighting parameter $\lambda_{\boldsymbol{G}} = 0.1$ was chosen for the proposed algorithm. Table 6.2 depicts the resulting PSNR values. Evidentially, the proposed algorithm appears to perform better than bicubic interpolation and the classical DIP formulation.

6.6. Synthesized Sequences

Let us now investigate if the proposed algorithm can be used as a post-processing technique to enhance the dynamic texture synthesis results produced by the DVAE framework presented in Chapter 5.

As with the synthesis experiments in Chapter 5, it is not possible to evaluate pixel-by-pixel error measures, since no high-resolution ground truth exists for synthetic video sequences. Therefore, the FID metric using the implementation in [110] once again was employed due to the lack of a better metric.

	Bicubic	DIP	Proposed
Flowing Water	32.17	**33.51**	33.10
Boiling Water	42.03	41.71	**43.03**
Sea	44.4	44.51	**45.14**
River	32.83	**32.85**	32.73
Mountain Stream	36.45	35.29	**36.5**
Spring Water	**37.7**	36.1	37.5
Fountain	**37.56**	36.5	37.25
Waterfall	**38.17**	36.76	38.11
Washing Machine	44.34	42.94	**45.31**
Flashing Lights	45.13	42.94	**45.59**
Fire Pot	47.6	43.53	**47.98**
Avg.	40.23	39.32	**40.57**

Table 6.2.: PSNR values for dynamic textures

For each dynamic texture, the FID score was computed between the frames from the training sequence to the first 9 frames of the according synthesized sequences that have then been subjected to one of the following three post-processing steps:

- No super-resolution,
- DIP-based super-resolution,
- Proposed super-resolution.

The employed FID implementation [110] automatically resizes images to fit the resolution 299×299, using bilinear interpolation. Such methods are often used for zooming in common media player software.

Same parameters as for the dynamic texture experiment in the previous section were chosen. In particular, the magnification factor 2 and a box filter were used. The resulting FID scores are listed in Table 6.3. In most of the tested sequences (7 out of 11), the proposed method produces the best FID scores. This is an indicator that the proposed super-resolution technique can help enhancing the visual quality of visual process synthesis, using the DVAE.

One should note that given the small number of sequences and the limited capability of the FID score to evaluate visual quality, the results may be not conclusive enough to recommend the described approach for super-resolution of synthesized visual process sequences. Particularly the fact that optimizing a neural network is a considerable drawback in comparison to quick and simple interpolation raises the demand for a significant advantage with respect to the visual quality of the resulted sequences. Whether or not the favorable results in Table 6.3 are significant is up for interpretation. For instance, one might ask, if using the proposed method actually

	w/o SR	DIP	Proposed
Flowing Water	143.4	130.6	**116.7**
Boiling Water	201.4	188.0	**185.3**
Sea	96.7	144.1	**92.0**
River	**124.8**	138.5	125.7
Mountain Stream	231.1	227.2	**190.1**
Spring Water	264.7	268.8	**247.5**
Fountain	**151.8**	162.5	152.8
Waterfall	**351.8**	357.6	363.8
Washing Machine	152.5	299.5	**147.5**
Flashing Lights	128.6	140.9	**123.8**
Fire Pot	93.7	**91.2**	103.7

Table 6.3.: FID scores

is advantageous in terms of the FID score, or if the limited set of sequences in Table 6.3 just happens to contain examples that make it perform better. A simple χ^2 test with the null hypothesis

> H_0 : Each one of the three tested approaches is equally likely to produce the best FID score.

would reject H_0 for $\alpha = 10\%$, but not for $\alpha = 5\%$. This means, that we cannot rule out the possibility that every algorithm is equally likely to perform best among the three with a confidence of higher than or equal to 95%. While the results in Table 6.3 are an indication that using the proposed algorithm for super-resolution of synthesized sequences can make sense, it may thus be sensible to decide on a case-by-case basis, using visual inspection. Figure 6.3 depicts the visual results of a frame from one of the evaluated sequences. While differences between the three algorithms are not particularly striking, the proposed method appears to produce results with more variation in color and luminosity which can be seen as an indication that more details are restored. In Figure 6.4 on the other hand, the difference in definition is less visible. At the same time, it becomes apparent that the proposed algorithm tends to introduce block artifacts. This can be explained by the fact that the underlying assumption of Eq. (6.5) does not hold here, because the frames have not been produced by downsizing a high-resolution image.

It appears thus that while the proposed algorithm can improve the quality of texture synthesis results for some sequences, it is less effective for others and simple interpolation methods may be preferable.

Appendix C.2 provides more visual results of the described experiments.

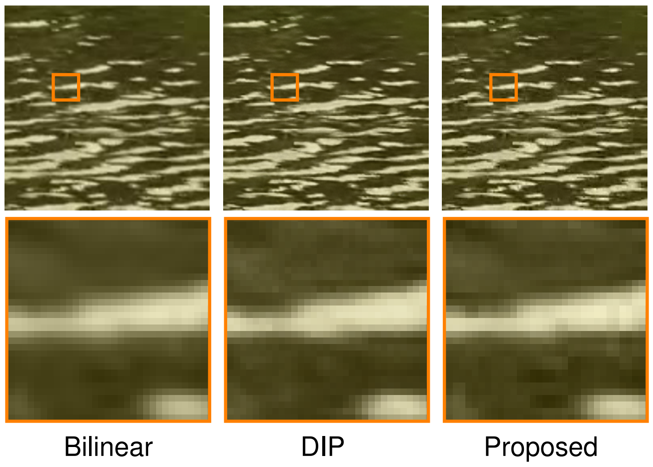


Figure 6.3.: Super-resolution results for *Flowing Water*

6.7. Discussion

DIP was originally introduced as a method that exploits the power of CNNs without relying on external datasets for training. By doing so, it has demonstrated a remarkable performance gain over classical methods that do not rely on learning. In this chapter, a super-resolution method based on DIP has been introduced that retains these advantages, but is capable to further improve the performance on common still-image super-resolution benchmarks. Likewise, it shows convincing results for the super-resolution of visual process videos.

Experimentally, this chapter looks into the question, if such algorithms are of any help to enhance the visual quality of video synthesis results, for instance the ones produced by the DVAE model introduced in Chapter 5. The motivation is grounded in the observation that video synthesis algorithms are rarely capable of producing visually appealing results in a high resolution by themselves.

Experimental evaluation in Section 6.6 indicates that such post-processing can indeed help to enhance the visual quality of synthesized video sequences. The FID score between the video frames processed by the proposed algorithms and the frames of the training sequence tends to be lower than, for instance, the FID score between interpolated video frames and the training sequence. Furthermore, some example sequences appear to contain more details when using the proposed method.

On the other hand, the visual improvement is not always striking and for some images, super-resolution results exhibit block artifacts. This chapter should thus not be interpreted as a unequivocal recommendation of the described algorithm for the post-processing of video synthesis results, although post-processing in itself still

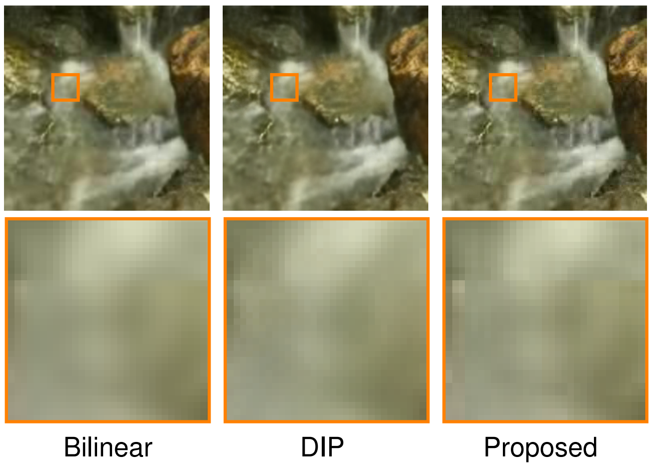


Figure 6.4.: Super-resolution results for *Spring Water*

might be of some use for enhancing the visual quality of video synthesis algorithms. One may recall LapSRN, for instance, which according to Table 6.1 yields impressive results for still-image super-resolution. Such performance gains are possible because it is trained on external datasets, unlike the proposed method. In the context of video synthesis, this could be used to our advantage, e.g. by training the model on the original high-resolution ground truth sequence. However, this would introduce additional computational load to the already demanding problem of synthesizing and post-processing visual processes. In the context of this thesis, such learning-based models have not been studied and it would go beyond the scope to evaluate the performance of such models in post-processing of synthesized video sequences.

7. Conclusion

This thesis discusses state space models for visual processes, in which the latent states follow a dynamic described by linear state transitions with Gaussian noise, and the observation results from a non-linear mapping of the latent space to the high-dimensional observation space. At the core of the research questions are methods to infer such models from finite, observed video sequences produced by some visual process. This entails the challenge of finding the right representation for the visual process observations such that they can be interpreted as latent space states produced by an autoregressive process. In addition to that, complementary tasks are also investigated, such as feature extraction from video frames, definition of distance measures, or post-processing via super-resolution.

First, the application of dynamic scene and dynamic texture recognition via kernels is investigated. To do so, the video sequences are subjected to a frame-wise feature extraction algorithm that computes histogram representations from each frame. These representations are then subsequently used to compute semi-linear dynamic models of the visual processes in question. The choice of an appropriate histogram representation, the right kernel and the distance measure on the computed model parameters is investigated and state-of-the-art performance on dynamic texture classification is demonstrated. The proposed features and distance measure have emphasized appearance over dynamics, which is also why the methods were evaluated on visual processes where appearance is the most distinctive aspect. While the proposed models are also theoretically applicable to videos of human actions to name an example, they would require some adaptation in order to shift the emphasis towards dynamics.

Second, a method to compute a generative model of visual processes, using a VAE, and a linear building block, the *dynamic layer* is introduced. The proposed architecture can be used to learn simultaneously the autoregressive state transition parameters, as well as the non-linear observation function that can be then used for synthesis purposes. Assessments based on visual observation and the *Fréchet Inception Distance* indicate that the proposed model can compete with state-of-the-art synthesis methods for visual processes, while maintaining a mathematically simple form. However, the resulting video sequences are of low resolution.

This poses the question if anything can be done about the visual quality once the sequence is synthesized. To investigate this possibility, a method for super-resolution of images is introduced and its performance on synthesized video sequences is evaluated. While the results do not prove that the proposed method generally improves the quality of a synthesized visual process sequence, they do suggest that post-processing can be used to improve the visual quality of video synthesis.

To summarize, the research in this thesis has demonstrated that semi-linear state space models inferred using methods from representation learning are a powerful method to describe visual processes. It has shown how to apply such models to recognition and synthesis tasks while emphasizing that such applications entail complementary problems that go beyond model inference, like feature extraction or post-processing.

One major obstacle in developing dynamic models that has become apparent in the course of the research in this thesis is the lack of benchmarks and evaluation metrics. In fact, this is a major limiting factor in studying visual process models. Even though this is less true for classification or segmentation tasks than it is for purely generative problems, such as synthesis, the development of widely accepted ways to measure the quality of a generative visual process model should not be underestimated in research during the years to come. So far, research in this area heavily relies on crowdsourcing platforms such as *Amazon Mechanical Turk*[1] to evaluate the quality of generated video sequences [123]. In the future, we can expect the demand for cheaper and more reproducible evaluation protocols to grow as a consequence of increased interest in deep generative modeling. Partially, this is due to applications such as data augmentation or simulation. Additionally, new trends in machine learning research, such as *causal machine learning* [107], require possibilities to model entire distributions of high-dimensional data. For visual process research to continue producing impactful results, it is inevitable to develop objective and widely applicable evaluation metrics for generative models.

The lack of training data is also an important limiting factor, although this has started to change. Typical visual process datasets such as YUPENN [34] are limited to only a handful of videos categorized into a few classes. This restricts the range of application for visual process models and is an obstacle for several visual process problems that requires domain specific data. For instance, identifying anomalous events in road traffic requires training sequences from the same or similar road scenes. Shortage in data is therefore a considerable bottleneck in visual process research and future advances will depend on how well this bottleneck will be overcome. Generative models, such as the proposed Dynamic VAE can help mitigating this obstacle, though further research is required to make such models capable of producing realistic and sufficiently diverse video data. Recent datasets such as the *Kinetics-700 Human Action Dataset* [23] show significant progress in gathering data for solving visual process related problems. However, visual process sequences require even higher dimensions than still images to be represented, implying that the curse of dimensionality cannot be entirely mitigated by using larger

[1] `https://www.mturk.com`

datasets. Instead, additional assumptions about data co-dependencies need to be introduced, which reduces data redundancies. In the context of this thesis, this has been done by making the assumption of a semi-linear process model. Using such a model, video sequences of arbitrary length can be stored using a compact parameter representation.

Overall, one can expect visual processes to stay relevant for research and development of visual intelligent systems. The results presented in this thesis lay out the foundation to tackle visual process problems in an interpretable, robust and versatile manner. Beyond that, the presented approaches to model stochastic processes as a semi-linear state space model using representation learning could also be useful for other kinds of temporal data such as in medical records or non-visual sensor data in settings. On a more theoretical level, it can be also worth investigating to what extent semi-linear dynamic systems can function as a mathematically simple and explainable approximation of more complex temporal models based on neural networks. In this context, the results in Chapter 5 can be potentially useful as they demonstrate how to employ neural networks for complex temporal behavior, without relying on computationally intractable architectures such as RNNs.

A. Alignment Distance Computation

Some aspects of the Alignment distance computation have been omitted in Chapter 3, since they constitute results that have been previously discussed [61, 111] in some form or another, and are thus not considered scientific contributions of their own. They are discussed in the following sections, as they still might be helpful in understanding the technical details of the framework in Chapter 3.

A.1. Computation of Bi-quadratic Coefficients

Let us consider the loss function to compute the Alignment distance in Section 3.2. When we assume that its argument is a Givens rotation rather than any orthogonal matrix, we get

$$\begin{aligned}&\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}\left(\Xi_1, \boldsymbol{G}_{k,l}(c,s) \bullet \Xi_2\right)\\ =&\operatorname{tr}(\boldsymbol{R}_1^\top \kappa(\boldsymbol{Y}_1,\boldsymbol{Y}_2)\boldsymbol{R}_2\boldsymbol{G}_{k,l}(c,s)^\top) + \lambda_{\boldsymbol{A}}\operatorname{tr}(\boldsymbol{A}_1^\top \boldsymbol{G}_{k,l}(c,s)\boldsymbol{A}_2\boldsymbol{G}_{k,l}(c,s)^\top)\\ &+\lambda_{\boldsymbol{B}}\operatorname{tr}(\boldsymbol{B}_1^\top \boldsymbol{G}_{k,l}(c,s)\boldsymbol{B}_2).\end{aligned} \tag{A.1}$$

By defining

$$\boldsymbol{D} = \lambda_{\boldsymbol{B}}\boldsymbol{B}_2\boldsymbol{B}_1^\top + \boldsymbol{R}_2^\top \kappa(\boldsymbol{Y}_2,\boldsymbol{Y}_1)\boldsymbol{R}_1, \tag{A.2}$$

we can rewrite the loss function as

$$\begin{aligned}&\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}\left(\Xi_1, \boldsymbol{G}_{k,l}(c,s) \bullet \Xi_2\right)\\ =&\lambda_{\boldsymbol{A}}\operatorname{tr}(\boldsymbol{A}_1^\top \boldsymbol{G}_{k,l}(c,s)\boldsymbol{A}_2\boldsymbol{G}_{k,l}(c,s)^\top) + \operatorname{tr}(\boldsymbol{G}_{k,l}(c,s)\boldsymbol{D}).\end{aligned} \tag{A.3}$$

This is a quadratic function in c and s. To compute its coefficients, observe that the two equations

$$\operatorname{tr}(\boldsymbol{G}_{k,l}(c,s)\boldsymbol{D}) = c(\boldsymbol{D})_{k,k} + s(\boldsymbol{D})_{l,k} - s(\boldsymbol{D})_{k,l} + c(\boldsymbol{D})_{l,l} + \sum_{\substack{i=1\ldots,n\\ i\neq k,l}} (\boldsymbol{D})_{i,i}, \tag{A.4}$$

and

$$\begin{aligned}
&\operatorname{tr}(\boldsymbol{A}_1^\top \boldsymbol{G}_{k,l}(c,s)\boldsymbol{A}_2\boldsymbol{G}_{k,l}(c,s)^\top)\\
={}&\sum_{\substack{i=1\ldots,n\\ i\neq k,l}} (c(\boldsymbol{A}_1)_{k,i} - s(\boldsymbol{A}_1)_{l,i})(\boldsymbol{A}_2)_{k,i} + (c(\boldsymbol{A}_1)_{l,i} + s(\boldsymbol{A}_1)_{k,i})(\boldsymbol{A}_2)_{l,i}\\
&+\sum_{\substack{i=1\ldots,n\\ i\neq k,l}} (\boldsymbol{A}_1)_{i,k}(c(\boldsymbol{A}_2)_{i,k} + s(\boldsymbol{A}_2)_{i,l}) + (\boldsymbol{A}_1)_{i,l}(c(\boldsymbol{A}_2)_{i,l} - s(\boldsymbol{A}_2)_{i,k})\\
&+(c(\boldsymbol{A}_1)_{k,k} - s(\boldsymbol{A}_1)_{l,k})(c(\boldsymbol{A}_2)_{k,k} + s(\boldsymbol{A}_2)_{k,l})\\
&+(c(\boldsymbol{A}_1)_{k,l} - s(\boldsymbol{A}_1)_{l,l})(c(\boldsymbol{A}_2)_{k,l} - s(\boldsymbol{A}_2)_{k,k})\\
&+(c(\boldsymbol{A}_1)_{l,k} + s(\boldsymbol{A}_1)_{k,k})(c(\boldsymbol{A}_2)_{l,k} + s(\boldsymbol{A}_2)_{l,l})\\
&+(c(\boldsymbol{A}_1)_{l,l} + s(\boldsymbol{A}_1)_{k,l})(c(\boldsymbol{A}_2)_{l,l} - s(\boldsymbol{A}_2)_{l,k})\\
&+\sum_{\substack{i=1\ldots,n\\ i\neq k,l}}\sum_{\substack{j=1\ldots,n\\ j\neq k,l}} (\boldsymbol{A}_1)_{k,l}(\boldsymbol{A}_2)_{k,l}.
\end{aligned} \tag{A.5}$$

hold. Assuming the form

$$\rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{G}_{k,l}(c,s)\bullet\Xi_2) = k_0c^2 + k_1cs + k_2s^2 + k_3c + k_4s + k_5, \tag{A.6}$$

The first three coefficients k_0, k_1, k_2 corresponding to second-order terms can be obtained by multiplying out the fourth to seventh line of Eq. (A.5), which yields

$$\begin{aligned}
k_0 ={}&\lambda_{\boldsymbol{A}}((\boldsymbol{A}_1)_{k,k}(\boldsymbol{A}_2)_{k,k} + (\boldsymbol{A}_1)_{k,l}(\boldsymbol{A}_2)_{k,l}\\
&+(\boldsymbol{A}_1)_{l,k}(\boldsymbol{A}_2)_{l,k} + (\boldsymbol{A}_1)_{l,l}(\boldsymbol{A}_2)_{l,l}),\\
k_1 ={}&\lambda_{\boldsymbol{A}}((\boldsymbol{A}_1)_{k,k}(\boldsymbol{A}_2)_{k,l} - (\boldsymbol{A}_1)_{l,k}(\boldsymbol{A}_2)_{k,k}\\
&-(\boldsymbol{A}_1)_{k,l}(\boldsymbol{A}_2)_{k,k} - (\boldsymbol{A}_1)_{l,l}(\boldsymbol{A}_2)_{k,l}\\
&+(\boldsymbol{A}_1)_{l,k}(\boldsymbol{A}_2)_{l,l} + (\boldsymbol{A}_1)_{k,k}(\boldsymbol{A}_2)_{l,k}\\
&-(\boldsymbol{A}_1)_{l,l}(\boldsymbol{A}_2)_{l,k} + (\boldsymbol{A}_1)_{k,l}(\boldsymbol{A}_2)_{l,l}),\\
k_2 ={}&-\lambda_{\boldsymbol{A}}((\boldsymbol{A}_1)_{l,k}(\boldsymbol{A}_2)_{k,l} - (\boldsymbol{A}_1)_{l,l}(\boldsymbol{A}_2)_{k,k}\\
&-(\boldsymbol{A}_1)_{k,k}(\boldsymbol{A}_2)_{l,l} + (\boldsymbol{A}_1)_{k,l}(\boldsymbol{A}_2)_{l,k}).
\end{aligned} \tag{A.7}$$

Likewise, multiplying out the second and third line of Eq. (A.5) and adding the respective first-order coefficients from Eq. (A.4) yields

$$\begin{aligned}
k_3 ={}&(\boldsymbol{D})_{k,k} + (\boldsymbol{D})_{l,l} + \lambda_{\boldsymbol{A}}\sum_{\substack{i=1\ldots,n\\ i\neq k,l}} ((\boldsymbol{A}_1)_{k,i}(\boldsymbol{A}_2)_{k,i}) + (\boldsymbol{A}_1)_{l,i}(\boldsymbol{A}_2)_{l,i})\\
&+\lambda_{\boldsymbol{A}}\sum_{\substack{i=1\ldots,n\\ i\neq k,l}} ((\boldsymbol{A}_1)_{i,k}(\boldsymbol{A}_2)_{i,k}) + (\boldsymbol{A}_1)_{i,l}(\boldsymbol{A}_2)_{i,l}),
\end{aligned} \tag{A.8}$$

$$\begin{aligned}k_4 = &- (\boldsymbol{D})_{k,l} + (\boldsymbol{D})_{l,k} + \lambda_{\boldsymbol{A}} \sum_{\substack{i=1\dots,n\\ i\neq k,l}} (-(\boldsymbol{A}_1)_{l,i}(\boldsymbol{A}_2)_{k,i}) + (\boldsymbol{A}_1)_{k,i}(\boldsymbol{A}_2)_{l,i})\\ &+ \lambda_{\boldsymbol{A}} \sum_{\substack{i=1\dots,n\\ i\neq k,l}} ((\boldsymbol{A}_1)_{i,k}(\boldsymbol{A}_2)_{i,l}) - (\boldsymbol{A}_1)_{i,l}(\boldsymbol{A}_2)_{i,k}).\end{aligned} \tag{A.9}$$

Finally adding together the constant terms from Eq. (A.4) and Eq. (A.5) yields

$$k_5 = \sum_{\substack{i=1\dots,n\\ i\neq k,l}} (\boldsymbol{D})_{i,i} + \sum_{\substack{i=1\dots,n\\ i\neq k,l}} \sum_{\substack{j=1\dots,n\\ j\neq k,l}} (\boldsymbol{A}_1)_{k,l}(\boldsymbol{A}_2)_{k,l}. \tag{A.10}$$

A.2. Constructing the Quartic Polynomial

The aim is to find the parameter that optimizes the function

$$h(c) = k_0c^2 \pm k_1c\sqrt{1-c^2} + k_2(1-c^2) + k_3c \pm k_4\sqrt{1-c^2} + k_5. \tag{A.11}$$

By taking the derivative, we obtain

$$\begin{aligned}h'(c) = \frac{dh}{dc} &= 2(k_0-k_2)c \pm k_1\frac{-2c^2+1}{\sqrt{1-c^2}} + k_3 \pm k_4\frac{-c}{\sqrt{1-c^2}}\\ &= 2(k_0-k_2)c + k_3 \mp \frac{1}{\sqrt{1-c^2}}\left(k_1(2c^2-1)+k_4c\right).\end{aligned} \tag{A.12}$$

As the next step, we want to find candidates for the solution $\hat{c}$ by observing the zeros of the derivative. Since any zero of $h'(c)$ is also a zero of any product containing $h'(c)$, we obtain

$$h'(c)\left(2(k_0-k_2)c + k_3 \pm \frac{1}{\sqrt{1-c^2}}\left(k_1(2c^2-1)+k_4c\right)\right) = 0. \tag{A.13}$$

Applying the third binomial formula yields

$$(2(k_0-k_2)c+k_3)^2 - \frac{1}{1-c^2}(k_1(2c^2-1)+k_4c)^2 = 0. \tag{A.14}$$

Multiplying by $(1-c^2)$ leads to

$$(1-c^2)(2(k_0-k_2)c+k_3)^2 - (k_1(2c^2-1)+k_4c)^2 = 0, \tag{A.15}$$

which can be multiplied out to obtain

$$\begin{aligned}&(1-c^2)(4(k_0-k_2)^2c^2 + 4k_3(k_0-k_2)c + k_3^2)\\ &\quad -(k_1^2(2c^2-1)^2 + 2k_1k_4(2c^3-c) + k_4^2c^2) = 0,\end{aligned} \tag{A.16}$$

and finally

$$\begin{aligned}-4((k_0-k_2)^2+k_1^2)c^4-4(k_3(k_0-k_2)+k_1k_4)c^3\\+(4(k_0-k_2)^2+4k_1^2-k_3^2-k_4^2)c^2+2(2k_3(k_0-k_2)+k_1k_4)c\\+(k_3^2-k_1^2)=0.\end{aligned} \tag{A.17}$$

Eq. (A.17) is a *quartic* equation for which closed form solutions exist [112].

A.3. Derivation of Convergence Properties

The proof of convergence properties of Algorithm 7 in Theorem 1 presented in the following builds upon the assumption that the value of a smooth function $L_{\text{align}}:\mathrm{O}(n)\to\mathbb{R}$ can always be increased (decreased) at a non-critical point $\boldsymbol{Q}$ by multiplying $\boldsymbol{Q}$ with a Givens rotation. Since this claim may not be self-explanatory, it is first proven in the following lemma, before revisiting Theorem 1.

Lemma 1. *Let*

$$L_{\text{align}}:\mathrm{O}(n)\to\mathbb{R} \tag{A.18}$$

be a smooth, real-valued function on the orthogonal group. Let $\boldsymbol{Q}$ be a non-critical point of L_{align}. Then we can find a Givens rotation $\boldsymbol{G}_{k,l}(c,s)$, such that

$$L_{\text{align}}(\boldsymbol{G}_{k,l}(c,s)\boldsymbol{Q})>L_{\text{align}}(\boldsymbol{Q}) \tag{A.19}$$

holds.

Proof. According to [1], the *tangent space* of $\mathrm{O}(n)$ at $\boldsymbol{Q}$ can be written as

$$\mathcal{T}_{\boldsymbol{Q}}(\mathrm{O}(n))=\{\boldsymbol{\Omega}\boldsymbol{Q}\mid\boldsymbol{\Omega}=-\boldsymbol{\Omega}^\top\}. \tag{A.20}$$

Let $\boldsymbol{T}_1,\dots,\boldsymbol{T}_{n(n-1)/2}$ be a basis for this space, i.e. $\operatorname{span}\{\boldsymbol{T}_1,\dots,\boldsymbol{T}_{n(n-1)/2}\}=\mathcal{T}_{\boldsymbol{Q}}(\mathrm{O}(n))$ and

$$\gamma_i:\mathbb{R}\to\mathrm{O}(n),\qquad i\in\{1,\dots,n(n-1)/2\} \tag{A.21}$$

be a smooth curve such that

$$\gamma_i(0)=\boldsymbol{Q} \tag{A.22}$$

and

$$\frac{d}{d\alpha}\gamma_i(\alpha)|_{\alpha=0}=\boldsymbol{T}_i \tag{A.23}$$

holds for any $i\in\{1,\dots,n(n-1)/2\}$. Then there is an $i'\in\{1,\dots,n(n-1)/2\}$ and an $\alpha'\in\mathbb{R}$ such that $L_{\text{align}}(\gamma_{i'}(\alpha'))>L_{\text{align}}(\boldsymbol{Q})$ holds.

It will now be shown that by fixing

$$i = n(k-1) - k(k+1)/2 + l, \tag{A.24}$$

with $k \in \{1, \dots, n-1\}, l \in \{k+1 \dots, n\}$, the curve

$$\gamma_i(\alpha) = \boldsymbol{G}_{k,l}(\cos(\alpha), \sin(\alpha))\boldsymbol{Q} \tag{A.25}$$

fulfills the aforementioned properties Eq. (A.22) and Eq. (A.23) for any $i \in \{1, \dots, n(n-1)/2\}$. It is easy to see that such a curve satisfies Eq. (A.22). Furthermore, we have

$$\frac{d}{d\alpha}\boldsymbol{G}_{k,l}(\cos(\alpha), \sin(\alpha))\boldsymbol{Q}|_{\alpha=0} = \boldsymbol{\Omega}_{k,l}\boldsymbol{Q}, \tag{A.26}$$

where $\boldsymbol{\Omega}_{k,l}$ is a skew-symmetric matrix with

$$(\boldsymbol{\Omega}_{k,l})_{\tilde{k},\tilde{l}} = \begin{cases} 1 & \text{if } \tilde{k} = k, \tilde{l} = l, \\ -1 & \text{if } \tilde{k} = l, \tilde{l} = k, \\ 0 & \text{otherwise.} \end{cases} \tag{A.27}$$

As such,

$$\left\{\frac{d}{d\alpha}\boldsymbol{G}_{k,l}(\cos(\alpha), \sin(\alpha))\boldsymbol{Q}|_{\alpha=0}\right\}_{k\in\{1,\dots,n-1\}, l\in\{k+1\dots,n\}} \tag{A.28}$$

is a basis of $\mathcal{T}_{\boldsymbol{Q}}(\mathrm{O}(n))$. This means that we can find a $k' \in \{1, \dots, n-1\}$, an $l' \in \{k+1 \dots, n\}$ and an $\alpha' \in \mathbb{R}$ such that

$$L_{\text{align}}(\boldsymbol{G}_{k',l'}(\cos(\alpha'), \sin(\alpha')\boldsymbol{Q})) > L_{\text{align}}(\boldsymbol{Q}) \tag{A.29}$$

holds. □

Lemma 1 states that at any point $\boldsymbol{Q}$, the tangent space of L_{align} can be spanned by a set of basis vectors, such that each one of these vectors can be written as

$$\frac{d}{d\alpha}\boldsymbol{G}_{k,l}(\cos(\alpha), \sin(\alpha))\boldsymbol{Q}|_{\alpha=0} \tag{A.30}$$

for some coordinate pair k, l.

This enables us to proof Theorem 1. Recall that it is stated as follows.

Theorem 1. *Let $(\boldsymbol{Q}^{(i)})_{i\in\mathbb{N}}$ be the sequence of orthogonal matrices generated by the* ***while*** *loop in Algorithm 7. It is bounded with respect to the Frobenius norm and thus has at least one accumulation point. Furthermore, every accumulation point of the sequence is almost certainly a critical point of the smooth objective function*

$$L_{\text{align}}(\boldsymbol{Q}) = \rho_{\lambda_{\boldsymbol{A}},\lambda_{\boldsymbol{B}}}(\Xi_1, \boldsymbol{Q} \cdot \Xi_2). \tag{A.31}$$

This statement can be proven as follows.

Proof. Since $\mathrm{O}(n)$ is bounded with regards to the Frobenius norm, so is any sequence of matrices in $\mathrm{O}(n)$. Thus $(\boldsymbol{Q}^{(i)})_{i\in\mathbb{N}}$ has at least one accumulation point.

The sequence $(L_{\text{align}}(\boldsymbol{Q}^{(i)}))_{i\in\mathbb{N}} \in \mathbb{R}$ is non-decreasing and bounded from above. Therefore it converges to a point $\hat{L} \in \mathbb{R}$ with $L_{\text{align}}(\boldsymbol{Q}^{(i)}) \leq \hat{L}\ \forall i \in \mathbb{N}$. Let $\hat{\boldsymbol{Q}}$ be an accumulation point of $(\boldsymbol{Q}^{(i)})_{i\in\mathbb{N}}$. Then, the equality $L_{\text{align}}(\hat{\boldsymbol{Q}}) = \hat{L}$ holds, and there is a monotonously increasing mapping

$$\eta : \mathbb{N} \to \mathbb{N}, \tag{A.32}$$

such that the sequence $(\boldsymbol{Q}^{(\eta(i))})_{i\in\mathbb{N}}$ converges to $\hat{\boldsymbol{Q}}$. Assume that $\hat{\boldsymbol{Q}}$ is not critical. According to Lemma 1, we can then find a Givens rotation $\boldsymbol{G}_{k,l}(c,s)$ with the property

$$L_{\text{align}}(\boldsymbol{G}_{k,l}(c,s)\hat{\boldsymbol{Q}}) > \hat{L}. \tag{A.33}$$

However, since the sequence $(\|\boldsymbol{Q}^{(\eta(i))} - \hat{\boldsymbol{Q}}\|_F)_{i\in\mathbb{N}}$ converges to zero, so does the sequence $(\|\boldsymbol{G}_{k,l}(c,s)\boldsymbol{Q}^{(\eta(i))} - \boldsymbol{G}_{k,l}(c,s)\hat{\boldsymbol{Q}}\|_F)_{i.\in\mathbb{N}}$, because the Frobenius norm is invariant with respect to orthogonal transformations. Therefore, due to the smoothness of L_{align}, there is an index $i' \in \mathbb{N}$, such that the following inequality holds for every $i \geq i'$.

$$L_{\text{align}}(\boldsymbol{G}_{k,l}(c,s)\boldsymbol{Q}^{(\eta(i))}) > \hat{L}. \tag{A.34}$$

Recall that at the beginning of each **while** iteration in Algorithm 7, the coordinates k, l are chosen randomly, which means that with a probability approaching 1, at an iteration $\eta' \geq \eta(i')$, k and l will be chosen such that there is a c and s that fulfill the inequality in Eq. (A.34). But that implies the inequality $L_{\text{align}}(\boldsymbol{Q}^{(\eta'+1)}) > \hat{L}$, because the objective function cannot be decreased by a **for** iteration of Algorithm 7. This is not possible, as $\hat{L}$ is an upper bound for the sequence $L_{\text{align}}(\boldsymbol{Q}^{(i)})_{i\in\mathbb{N}}$. □

In [111], a different proof for Theorem 1 is provided.

B. Possible Extensions of the DVAE

Generative models are often widely adaptable and applicable to a variety of use cases. This is also true for the proposed DVAE framework. Such adaptations usually require significant effort that involves deciding on appropriate architectural choices and hyperparameters, implementing them as software packages and evaluating on suitable datasets. Hence, Chapter 5 has focused on one particular application, namely video synthesis. To establish that the scientific contribution of Chapter 5 goes beyond the specific application scenario described there, this appendix discusses potential extensions of the proposed framework.

B.1. Application to Frame Prediction

A widely discussed application of generative dynamic process models is frame prediction. At each time t, Eq. (5.2a) yields an expected value of the latent state in the step $t+1$ as $\mathbb{E}_{\mathbf{v}_t}[\mathbf{x}_{t+1}] = \boldsymbol{A}\boldsymbol{x}_t$. Using the learned observer function $\Gamma_{\boldsymbol{\zeta}}$, the predicted frame at time $t+1$ can be then estimated as $\Gamma_{\boldsymbol{\zeta}}(\boldsymbol{A}\boldsymbol{x}_t)$, if $\boldsymbol{x}_t$ is known.

As a proof-of-concept result, Table B.1 depicts PSNR values for predicting the last frame for (the low-resolution version of) the respective dynamic texture sequence. The latent variable of the preceding frame is estimated using the VAE encoder. Modern frame prediction algorithms incorporate several heuristics with regards to the video at hand so that a prediction algorithm based on the Dynamic VAE might require additional pre- and post-processing in order to compete with contemporary approaches.

B.2. Generalization to Higher-order Markov Processes

Assumption 1 restricts the scope of Chapter 5 to first-order Markov processes with second order stationarity. This is in line with Eq. (5.2a) as the latent VAR model is obviously first-order Markov. For many applications this is sufficient, since Assumption 1 is an accurate characterization for typical publicly available benchmark datasets. However, this assumption does not mean that DVAEs cannot be applied to Markov processes of higher order. To show this, a procedure is described in the following that can be used to generalize DVAEs to processes with Markov order m and $m+1$-th order stationarity, where $m > 1$. The discussion serves to establish theoretical feasibility. Data from publicly available datasets are rarely suitable for

Sequence	LDS	DVAE
Flowing Water	**14.28**	14.08
Boiling Water	25.54	**27.67**
Sea	23.77	**26.86**
River	**18.09**	17.81
Mountain Stream	21.89	**24.16**
Spring Water	21.25	**22.06**
Fountain	20.04	**21.61**
Waterfall	**18.50**	18.34
Washing Machine	29.59	**33.23**
Flashing Lights	22.57	**39.36**
Fire Pot	**18.66**	16.72

Table B.1.: PSNR values for prediction of last frame

such complex modeling due to limited sequence length. Therefore, experimental evaluation is not carried out.

The general procedure is as follows.

1. First, an appropriate model for the joint probability of $m+1$ succeeding observations needs to be set up. This requires generalizing Eq. (5.18) to longer Markov chains. Since the distribution in question is always zero-mean Gaussian, it is entirely defined by its covariance matrix.

2. Next, a dynamic layer that performs a multiplication with a lower-triangular $(m+1) \times (m+1)$ block matrix $\boldsymbol{F}$ needs to be designed. The elements of this matrix are learned along with the VAE weights. The product $\boldsymbol{F}\boldsymbol{F}^\top$ must be (approximately) equal to the covariance matrix.

3. To make sure that the visual process model fulfills the imposed assumptions, regularizers need to be introduced. They ensure the Toeplitz structure of the covariance matrix.

4. Once the model including the elements of $\boldsymbol{F}$ is learned, a VAR model for the according Markov order can be computed.

To illustrate this procedure, the steps are described for $m+1$- stationary visual processes with the Markov order $m=2$, in the following.

1. Due to the stationarity assumption, the covariance matrix for three succeeding latent states $\mathbf{h}_t, \mathbf{h}_{t+1}, \mathbf{h}_{t+2}$ has the form

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} \boldsymbol{I}_n & \operatorname{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1}) & \operatorname{Cov}(\mathbf{h}_t, \mathbf{h}_{t+2}) \\ \operatorname{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1})^\top & \boldsymbol{I}_n & \operatorname{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1}) \\ \operatorname{Cov}(\mathbf{h}_t, \mathbf{h}_{t+2})^\top & \operatorname{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1})^\top & \boldsymbol{I}_n \end{bmatrix}. \tag{B.1}$$

2. The matrix describing the dynamic layer has the form

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{I}_n & & \\ \boldsymbol{F}_1 & \boldsymbol{F}_2 & \\ \boldsymbol{F}_3 & \boldsymbol{F}_4 & \boldsymbol{F}_5 \end{bmatrix}. \tag{B.2}$$

The outputs of the dynamic layer will thus have the distribution

$$p\left(\begin{bmatrix} \mathbf{h}_1^\top & \mathbf{h}_2^\top & \mathbf{h}_3^\top \end{bmatrix}^\top\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{h}_1^\top & \mathbf{h}_2^\top & \mathbf{h}_3^\top \end{bmatrix}^\top;\ 0, \boldsymbol{F}\boldsymbol{F}^\top\right). \tag{B.3}$$

We thus need to achieve

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{F}_1^\top & \boldsymbol{F}_3^\top \\ \boldsymbol{F}_1 & \boldsymbol{F}_1\boldsymbol{F}_1^\top + \boldsymbol{F}_2\boldsymbol{F}_2^\top & \boldsymbol{F}_1\boldsymbol{F}_3^\top + \boldsymbol{F}_2\boldsymbol{F}_4^\top \\ \boldsymbol{F}_3 & \boldsymbol{F}_3\boldsymbol{F}_1^\top + \boldsymbol{F}_4\boldsymbol{F}_2^\top & \boldsymbol{F}_3\boldsymbol{F}_3^\top + \boldsymbol{F}_4\boldsymbol{F}_4^\top + \boldsymbol{F}_5\boldsymbol{F}_5^\top \end{bmatrix}. \tag{B.4}$$

3. This can be enforced by using the regularizer

$$\begin{aligned} R(\boldsymbol{F}) =& \lambda_1 \|\boldsymbol{F}_1\boldsymbol{F}_1^\top + \boldsymbol{F}_2\boldsymbol{F}_2^\top - \boldsymbol{I}_n\|_F^2 \\ &+ \lambda_2 \|\boldsymbol{F}_3\boldsymbol{F}_3^\top + \boldsymbol{F}_4\boldsymbol{F}_4^\top + \boldsymbol{F}_5\boldsymbol{F}_5^\top - \boldsymbol{I}_n\|_F^2 \\ &+ \lambda_3 \|\boldsymbol{F}_3\boldsymbol{F}_1^\top + \boldsymbol{F}_4\boldsymbol{F}_2^\top - \boldsymbol{F}_1\|_F^2, \end{aligned} \tag{B.5}$$

with $\lambda_1, \lambda_2, \lambda_3 > 0$.

4. A second-order VAR model has the form

$$\boldsymbol{h}_{t+2} = \boldsymbol{A}_0\boldsymbol{h}_t + \boldsymbol{A}_1\boldsymbol{h}_{t+1} + \boldsymbol{B}\boldsymbol{v}_t,\ \mathbf{v}_t \sim \mathcal{N}(\mathbf{v}_t, 0, \boldsymbol{I}_n). \tag{B.6}$$

By our assumptions on the process, this yields the system of equations,

$$\begin{aligned} \mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_t) &= \boldsymbol{A}_0\boldsymbol{A}_0^\top + \boldsymbol{A}_1\boldsymbol{A}_1^\top + \boldsymbol{A}_0\mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1})\boldsymbol{A}_1^\top \\ &\quad + \boldsymbol{A}_1\mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1})^\top\boldsymbol{A}_0^\top + \boldsymbol{B}\boldsymbol{B}^\top \\ \mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1}) &= \mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1})^\top\boldsymbol{A}_0^\top + \boldsymbol{A}_1^\top, \\ \mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_{t+2}) &= \boldsymbol{A}_0^\top + \mathrm{Cov}(\mathbf{h}_t, \mathbf{h}_{t+1})\boldsymbol{A}_1^\top. \end{aligned} \tag{B.7}$$

Thus, by using Eq. (B.4), we can replace the covariance terms by $\boldsymbol{F}_1$, $\boldsymbol{F}_2$ and $\boldsymbol{I}_n$, respectively, which yields

$$\begin{aligned} \boldsymbol{F}_1^\top &= \boldsymbol{F}_1\boldsymbol{A}_0^\top + \boldsymbol{A}_1^\top, \\ \boldsymbol{F}_3 &= \boldsymbol{A}_0 + \boldsymbol{A}_1\boldsymbol{F}_1, \\ \boldsymbol{I}_n &= \boldsymbol{A}_0\boldsymbol{A}_0^\top + \boldsymbol{A}_1\boldsymbol{A}_1^\top + \boldsymbol{A}_0\boldsymbol{F}_1^\top\boldsymbol{A}_1^\top + \boldsymbol{A}_1\boldsymbol{F}_1\boldsymbol{A}_0^\top + \boldsymbol{B}\boldsymbol{B}^\top. \end{aligned} \tag{B.8}$$

This system of equations can be solved either numerically or analytically to obtain the VAR parameters $\boldsymbol{A}_0, \boldsymbol{A}_1, \boldsymbol{B}$.

B.3. Generative Adversarial Nets

The VAE is not the only generative framework that can be equipped with a dynamic layer. Since GANs can also be trained to map standard Gaussian noise to real-world images, they present a natural environment for learning models of visual processes by means of a dynamic layer. To this end, the architecture in Figure 5.1 is employed as the generator of the GAN.

Like the dynamic VAE, a “dynamic” GAN is trained with frame pairs generated from the observed sequence(s) and the discriminator evaluates if the frame pair at its input originated from the generator or the training set.

C. Visual Results

Visual results of the frameworks in Chapter 5 and Chapter 6 are presented in the following.

C.1. Synthesis Results of the DVAE

MNIST Results

Fig. C.1 - Fig. C.6 depict additional synthesis results of the MNIST experiment.

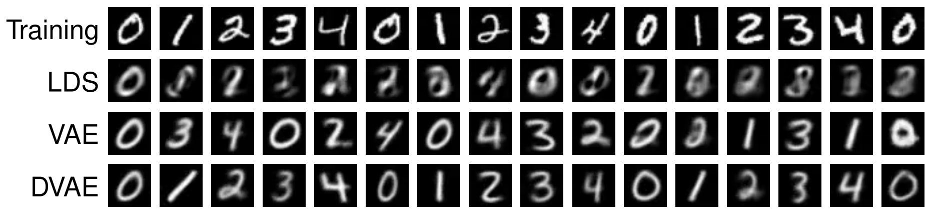


Figure C.1.: Synthesis of MNIST sequence 0123401234...

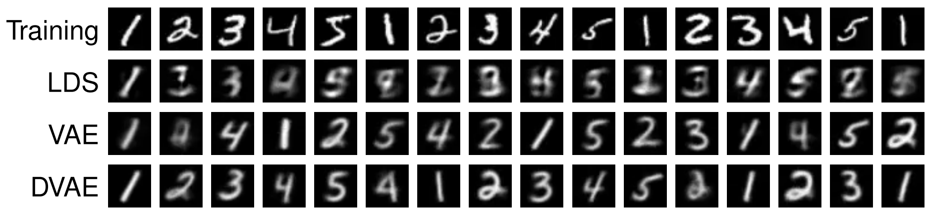


Figure C.2.: Synthesis of MNIST sequence 1234512345...

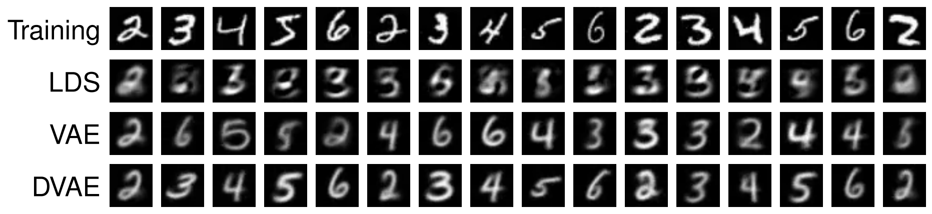


Figure C.3.: Synthesis of MNIST sequence 2345623456. . .

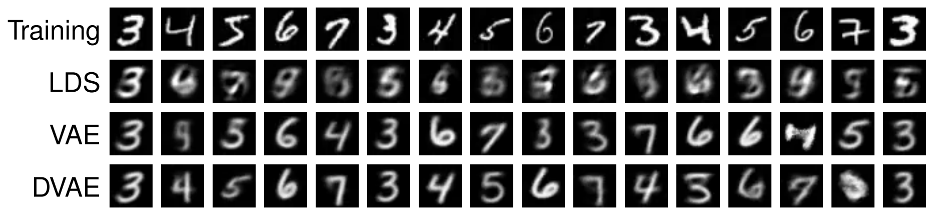


Figure C.4.: Synthesis of MNIST sequence 3456734567. . .

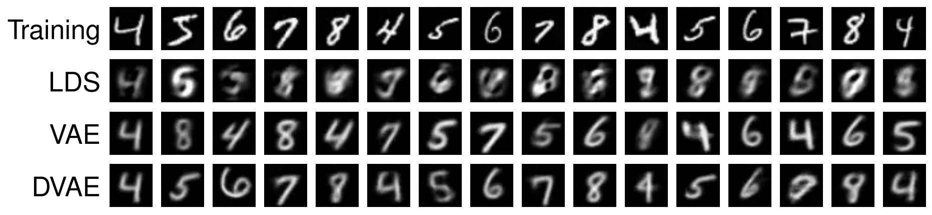


Figure C.5.: Synthesis of MNIST sequence 4567845678. . .

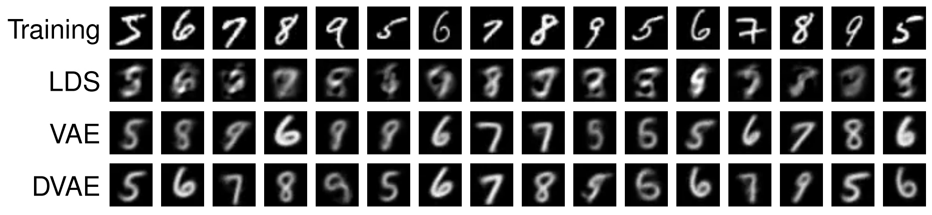


Figure C.6.: Synthesis of MNIST sequence 5678956789. . .

Small NORB Results

Fig. C.7 - Fig. C.11 depict additional synthesis results of the Small NORB experiment.

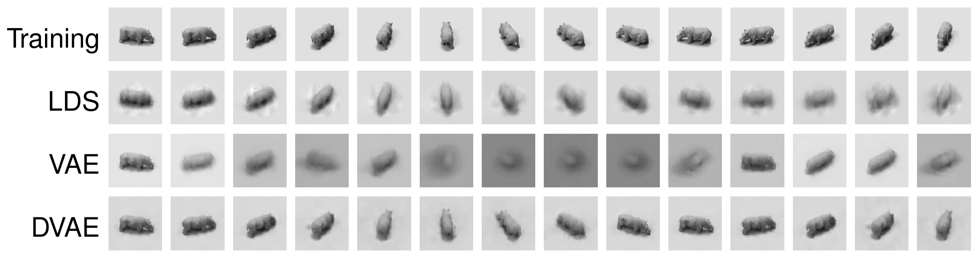


Figure C.7.: Synthesis of a rotation sequence of Small NORB images (Category 0, Instance 4)

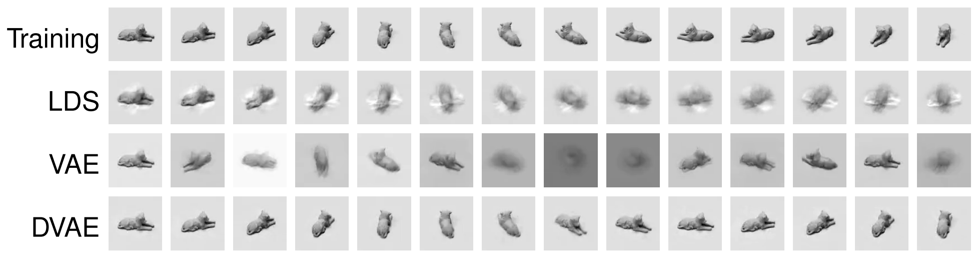


Figure C.8.: Synthesis of a rotation sequence of Small NORB images (Category 0, Instance 6)

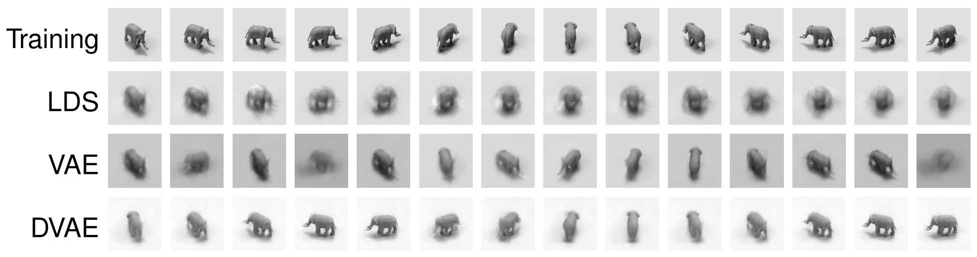


Figure C.9.: Synthesis of a rotation sequence of Small NORB images (Category 0, Instance 7)

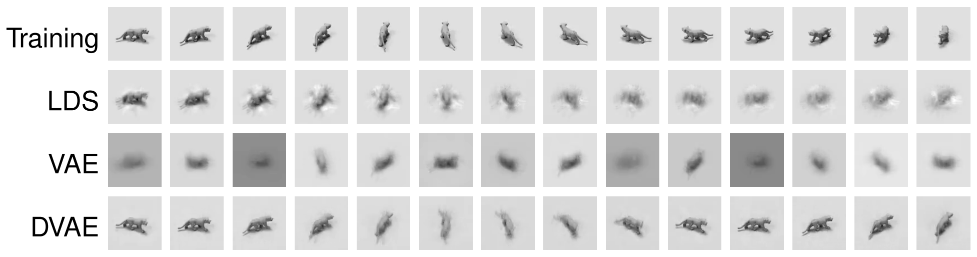


Figure C.10.: Synthesis of a rotation sequence of Small NORB images (Category 0, Instance 8)

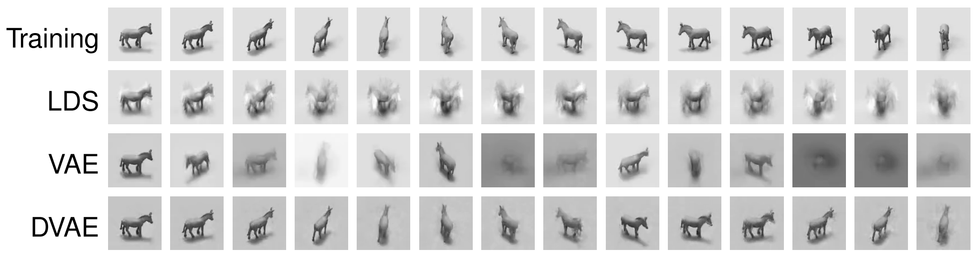


Figure C.11.: Synthesis of a rotation sequence of Small NORB images (Category 0, Instance 9)

Dynamic Textures

Fig. C.12 - Fig. C.21 depict additional synthesis results of dynamic textures. Note that the STGCONV and DGM frames were synthesized in a slightly higher resolution, 224×224.

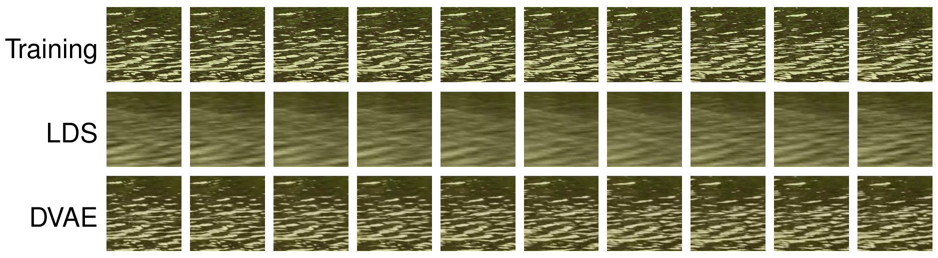


Figure C.12.: Synthesis of dynamic texture *Flowing Water*

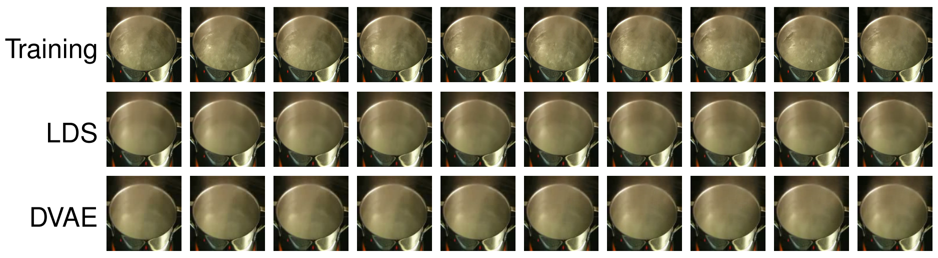


Figure C.13.: Synthesis of dynamic texture *Boiling Water*

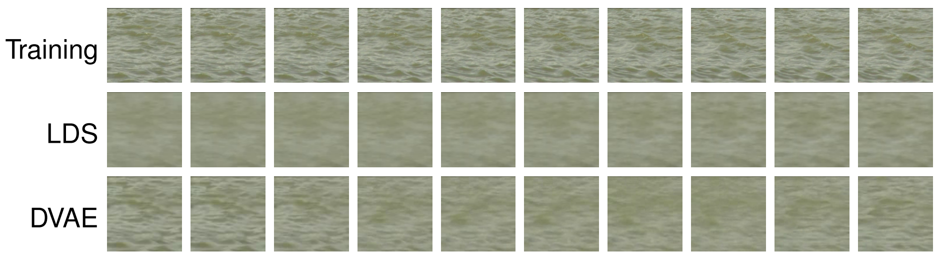


Figure C.14.: Synthesis of dynamic texture *Sea*

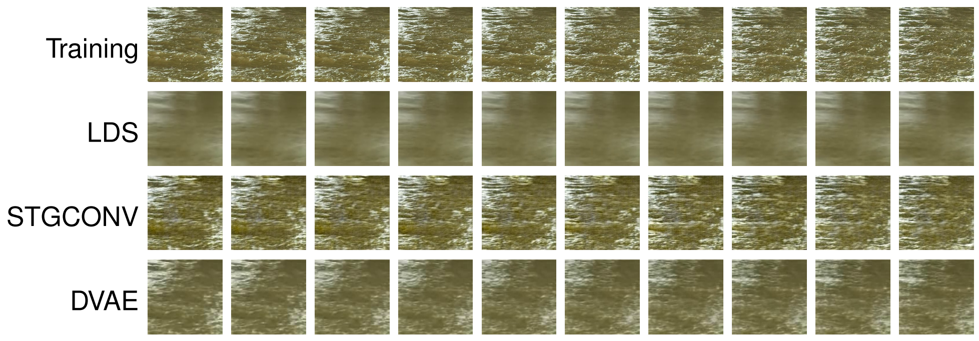


Figure C.15.: Synthesis of dynamic texture *River*

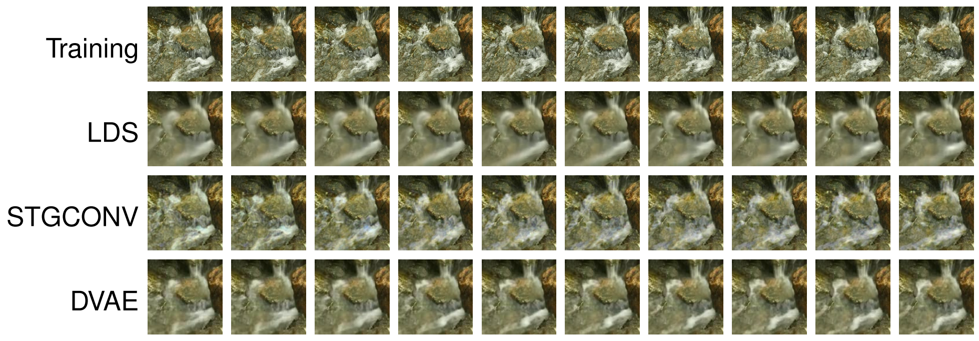


Figure C.16.: Synthesis of dynamic texture *Spring water*

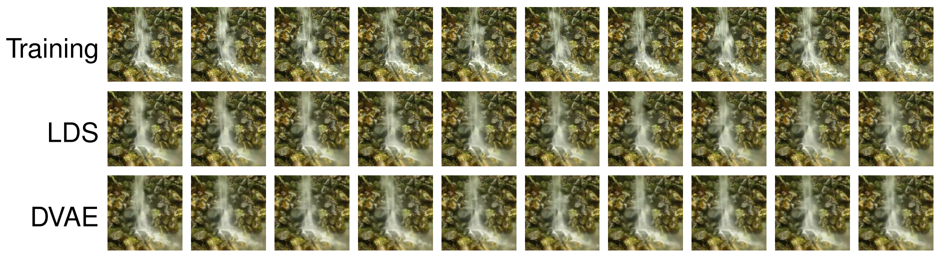


Figure C.17.: Synthesis of dynamic texture *Mountain Stream*

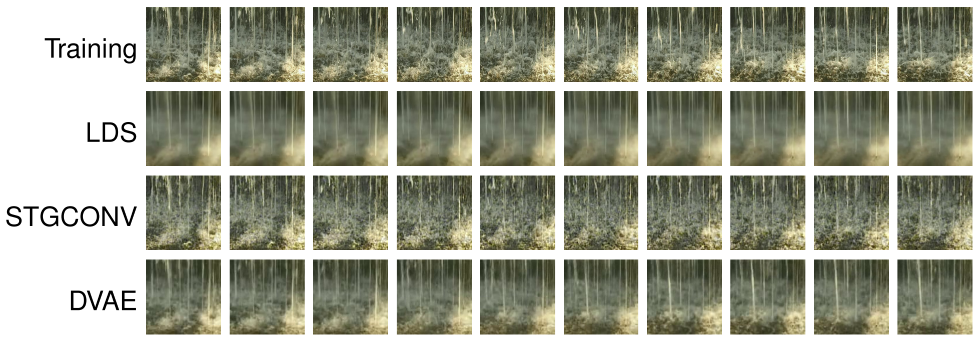


Figure C.18.: Synthesis of dynamic texture *Fountain*

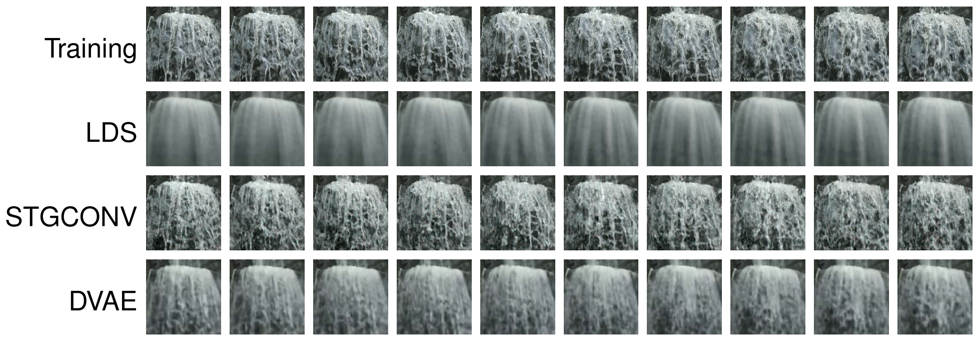


Figure C.19.: Synthesis of dynamic texture *Waterfall*

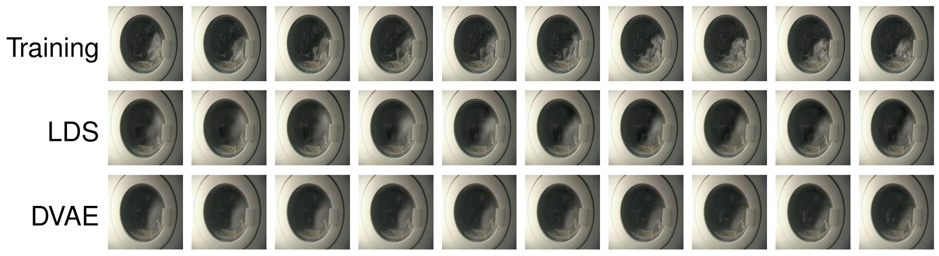


Figure C.20.: Synthesis of dynamic texture *Washing Machine*

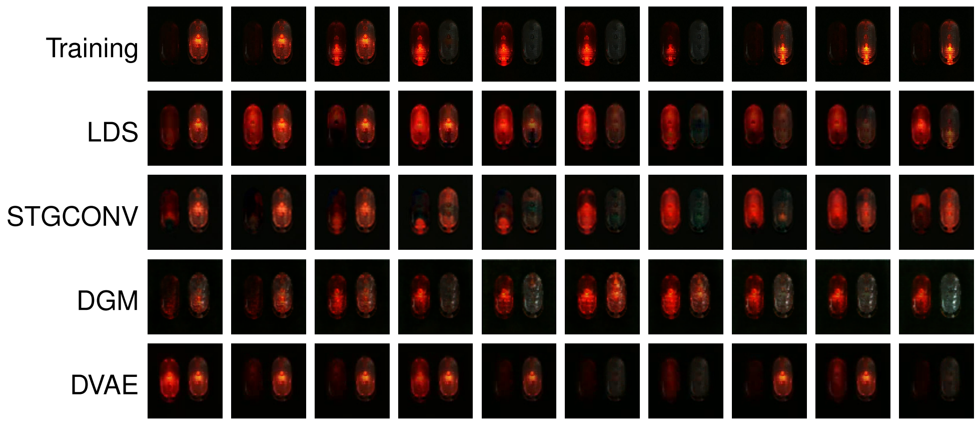


Figure C.21.: Synthesis of dynamic texture *Flashing lights*

C.2. Super-resolution Results

Figure C.22-Figure C.30 depict super-resolution results of the first synthesized frame for different visual processes. The proposed method is compared against the deep image prior and bilinear interpolation. For many dynamic textures the differences between the methods are hardly noticeable, e.g. *Washing Machine* and *Firepot*. Other dynamic textures, e.g. *Flashing Lights*, clearly benefit from the proposed approach. Yet some other sequences, e.g. *Fountain*, exhibit block artifacts for the proposed approach.

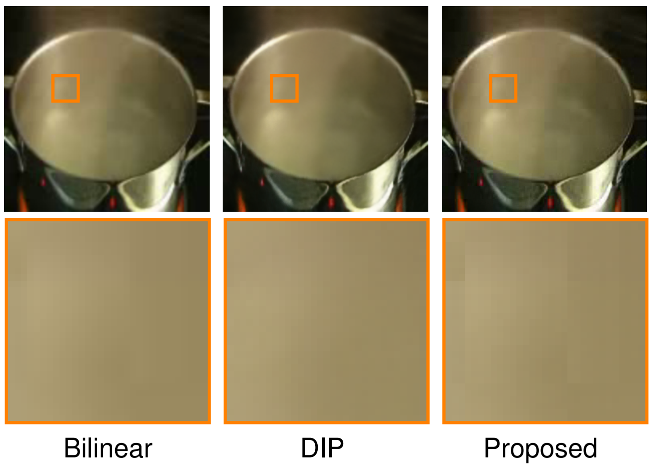


Figure C.22.: Super-resolution results for *Boiling Water*

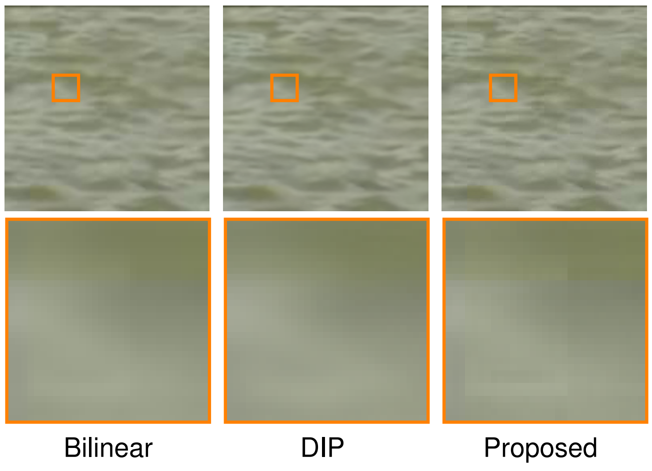


Figure C.23.: Super-resolution results for *Sea*

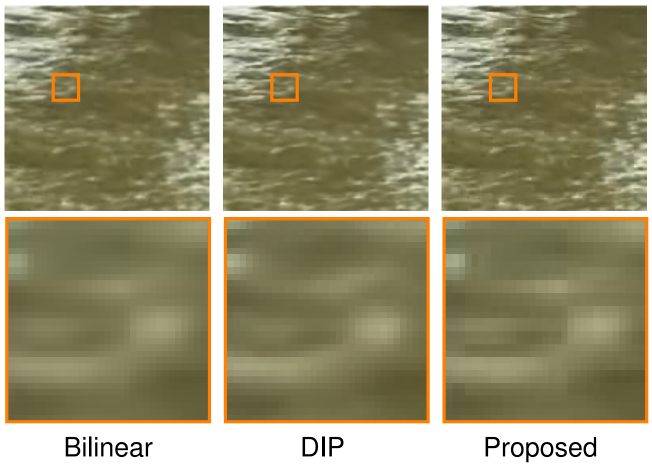


Figure C.24.: Super-resolution results for *River*

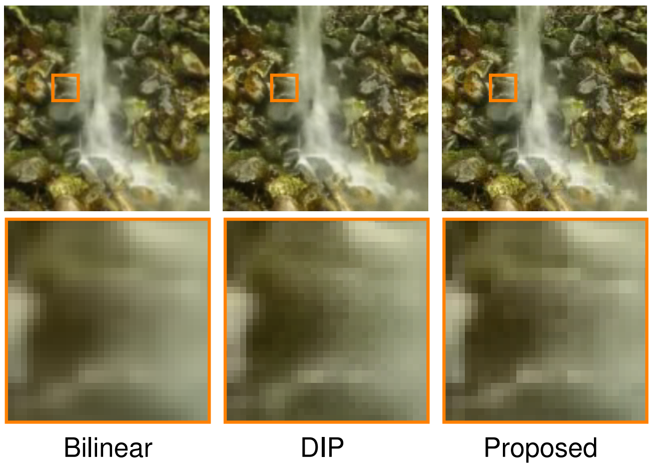


Figure C.25.: Super-resolution results for *Mountain Stream*

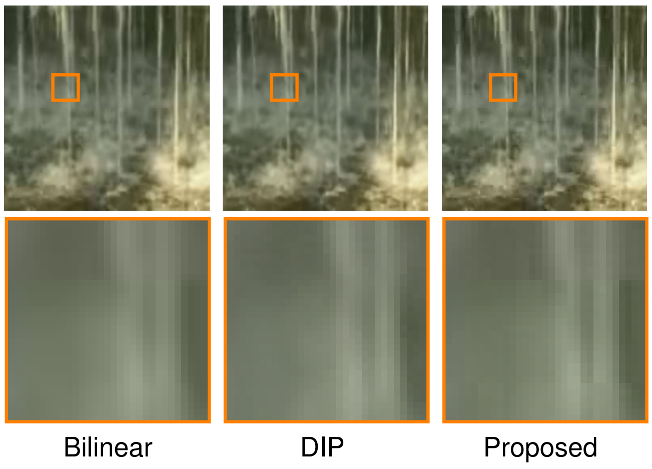


Figure C.26.: Super-resolution results for *Fountain*

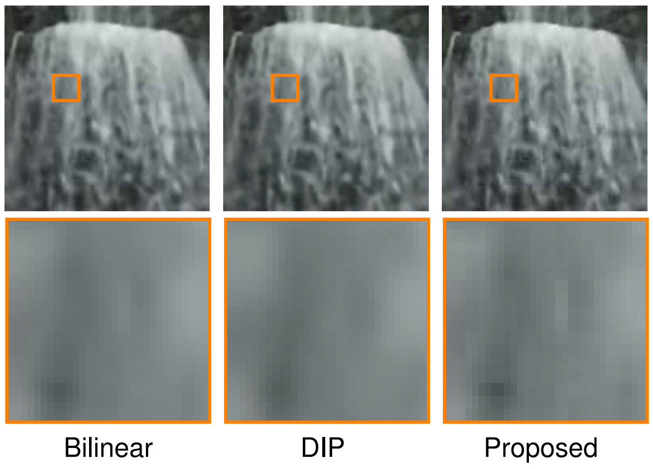


Figure C.27.: Super-resolution results for *Waterfall*

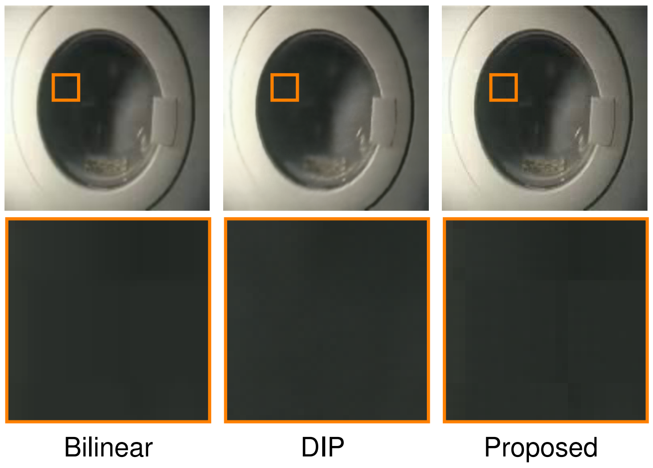


Figure C.28.: Super-resolution results for *Washing Machine*

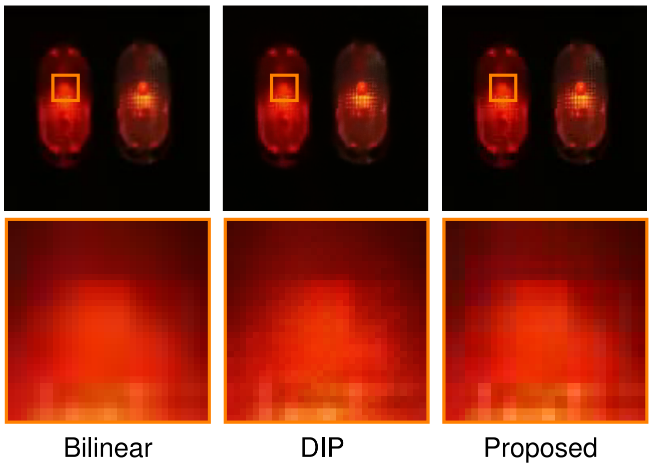


Figure C.29.: Super-resolution results for *Flashing Lights*

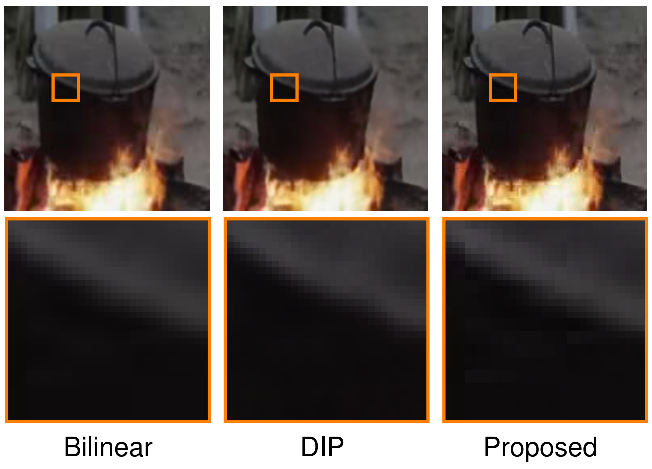


Figure C.30.: Super-resolution results for *Firepot*

Bibliography

[1] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] Bijan Afsari, Rizwan Chaudhry, Avinash Ravichandran, and René Vidal. Group action induced distances for averaging and clustering linear dynamical systems with applications to the analysis of dynamic scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2208–2215, 2012.

[3] Bijan Afsari and René Vidal. The alignment distance on spaces of linear dynamical systems. In *IEEE Annual Conference on Decision and Control (CDC)*, pages 1162–1167, 2013.

[4] Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.

[5] Vincent Andrearczyk and Paul F. Whelan. Convolutional neural network on three orthogonal planes for dynamic texture classification. *Pattern Recognition*, 76:36–49, 2018.

[6] Mathieu Andreux, Tomás Angles, Georgios Exarchakis, Roberto Leonarduzzi, Gaspar Rochette, Louis Thiry, John Zarka, Stéphane Mallat, Joakim Andén, Eugene Belilovsky, Joan Bruna, Vincent Lostanlen, Muawiz Chaudhary, Matthew J. Hirn, Edouard Oyallon, Sixin Zhang, Carmine Cella, and Michael Eickenberg. Kymatio: Scattering transforms in python. *Journal of Machine Learning Research*, 21(60):1–6, 2020.

[7] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3981–3989, 2016.

[8] Tomás Angles and Stéphane Mallat. Generative networks as inverse problems with scattering transforms. In *International Conference on Learning Representations (ICLR)*, 2018.

[9] Shervin R. Arashloo, Mehdi C. Amirani, and Ardeshir Noroozi. Dynamic texture representation using a deep multi-scale convolutional network. *Journal of Visual Communication and Image Representation*, 43:89–97, 2017.

[10] Shervin R. Arashloo and Josef Kittler. Dynamic texture recognition using multiscale binarized statistical image features. *IEEE Transactions on Multimedia*, 16:2099–2109, 2014.

[11] Yuksel A. Aslandogan and Clement T. Yu. Techniques and systems for image and video retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):56–63, 1999.

[12] Annalisa Barla, Francesca Odone, and Alessandro Verri. Histogram intersection kernel for image classification. In *IEEE International Conference on Image Processing (ICIP)*, pages III–513, 2003.

[13] Richard H. Bartels and George W. Stewart. Solution of the matrix equation ax + xb = c [f4]. *Communications of the ACM*, 15(9):820–826, 1972.

[14] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006.

[15] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961.

[16] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[17] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *British Machine Vision Conference (BMVC)*. BMVA press, 2012.

[18] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[19] Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1395–1402, 2005.

[20] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. In *International Conference on Machine Learning*, pages 600–609, 2018.

[21] Léon Bottou. Stochastic learning. In *Summer School on Machine Learning*, pages 146–168. Springer, 2003.

[22] Haoye Cai, Chunyan Bai, Yu-Wing Tai, and Chi-Keung Tang. Deep video generation, prediction and completion of human action sequences. In *European Conference on Computer Vision (ECCV)*, pages 366–382. Springer, 2018.

[23] Joao Carreira, Eric Noland, Chloe Hillier, and Andrew Zisserman. A short note on the kinetics-700 human action dataset. *arXiv preprint arXiv:1907.06987*, 2019.

[24] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[25] Antoni B. Chan and Nuno Vasconcelos. Probabilistic kernels for the classification of auto-regressive visual processes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 846–851, 2005.

[26] Antoni B. Chan and Nuno Vasconcelos. Classifying video with kernel dynamic textures. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–6. IEEE, 2007.

[27] Antoni B. Chan and Nuno Vasconcelos. Modeling, clustering, and segmenting video with mixtures of dynamic textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):909–926, 2008.

[28] Rizwan Chaudhry, Avinash Ravichandran, Gregory Hager, and René Vidal. Histograms of oriented optical flow and Binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1932–1939, 2009.

[29] Vedang Chauhan and Brian Surgenor. A comparative study of machine vision based methods for fault detection in an automated assembly machine. *Procedia Manufacturing*, 1:416–428, 2015.

[30] Hao Chen, Ruimin Hu, Dan Mao, Rui Zhong, and Zhongyuan Wang. Video coding using dynamic texture synthesis. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 203–208, 2010.

[31] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, 2005.

[32] Katrien De Cock and Bart De Moor. Subspace angles between arma models. *Systems & Control Letters*, 46(4):265–270, 2002.

[33] Lokenath Debnath and Piotr Mikusinski. *Introduction to Hilbert spaces with applications*. Academic press, 2005.

[34] Konstantinos G. Derpanis, Matthieu Lecce, Kostas Daniilidis, and Richard P. Wildes. Dynamic scene understanding: The role of orientation features in space and time in scene classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1306–1313, 2012.

[35] Kosmas Dimitropoulos, Panagiotis Barmpoutis, and Nikos Grammalidis. Spatio-temporal flame modeling and dynamic texture analysis for automatic video-based fire detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(2):339–351, 2015.

[36] Minh N. Do and Martin Vetterli. Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance. *IEEE Transactions on Image Processing*, 11(2):146–158, 2002.

[37] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[38] Gianfranco Doretto, Alessandro Chiuso, Ying Nian Wu, and Stefano Soatto. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, 2003.

[39] Gianfranco Doretto, Daniel Cremers, Paolo Favaro, and Stefano Soatto. Dynamic texture segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1236–1236, 2003.

[40] Petros Drineas and Michael W. Mahoney. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(12):2153–2175, 2005.

[41] Sloven Dubois, Renaud Péteri, and Michel Ménard. Characterization and recognition of dynamic textures based on the 2d+ t curvelet transform. *Signal, Image and Video Processing*, 9(4):819–830, 2015.

[42] Li Fei-Fei and Pietro Perona. A bayesian hierarchical model for learning natural scene categories. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 524–531, 2005.

[43] Fajwel Fogel, Irène Waldspurger, and Alexandre d'Aspremont. Phase retrieval for imaging problems. *Mathematical programming computation*, 8(3):311–335, 2016.

[44] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, 2004.

[45] Andel Früh. Spring-pendulum. `https://commons.wikimedia.org/wiki/File:Spring-pendulum.jpg`, License: `https://creativecommons.org/licenses/by-sa/3.0/legalcode`.

[46] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.

[47] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[48] Isma Hadji and Richard P. Wildes. A spatiotemporal oriented energy network for dynamic texture recognition. In *IEEE International Conference on Computer Vision (ICCV)*, pages 3066–3074, 2017.

[49] Robert M. Haralick and G.L. Kelly. Pattern recognition with measurement space and spatial clustering for multiple image. *Proceedings of the IEEE*, 57:654 – 665, 05 1969.

[50] Moritz Hardt, Tengyu Ma, and Benjamin Recht. Gradient descent learns linear dynamical systems. *The Journal of Machine Learning Research*, 19(1):1025–1068, 2018.

[51] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1664–1673, 2018.

[52] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[53] Elad Hazan, Karan Singh, and Cyril Zhang. Learning linear dynamical systems via spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6702–6712, 2017.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[55] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, page 6629–6640. Curran Associates, Inc., 2017.

[56] Sungeun Hong, Jongbin Ryu, and Hyun S. Yang. Not all frames are equal: aggregating salient features for dynamic texture classification. *Multidimensional Systems and Signal Processing*, (29):279–298, 2018.

[57] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[58] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *Graphical Models and Image Processing*, 53(3):231–239, 1991.

[59] Ylva Jansson and Tony Lindeberg. Dynamic texture recognition using time-causal and time-recursive spatio-temporal receptive fields. *Journal of Mathematical Imaging and Vision*, 60(9):1369–1398, 2018.

[60] Tony Jebara and Risi Kondor. Bhattacharyya and expected likelihood kernels. In *Learning theory and kernel machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2003.

[61] Nicolas D. Jimenez, Bijan Afsari, and René Vidal. Fast Jacobi-type algorithm for computing distances between linear dynamical systems. In *IEEE European Control Conference (ECC)*, pages 3682–3687, 2013.

[62] Matthew Johnson, David K. Duvenaud, Alex Wiltschko, Ryan P. Adams, and Sandeep R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2946–2954, 2016.

[63] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations (ICLR)*, 2017.

[64] Shiva Kaul. Linear dynamical systems as a core computational primitive. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[65] Mahmut Kaya and Hasan Sakir Bilge. Deep metric learning: A survey. *Symmetry*, 11(9), 2019.

[66] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[67] Manesh Kokare, Prabir K. Biswas, and Biswanath N. Chatterji. Rotation-invariant texture image retrieval using rotated complex wavelet filters. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(6):1273–1282, 2006.

[68] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[69] Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[70] Roland Kwitt and Andreas Uhl. Image similarity measurement by Kullback-Leibler divergences between complex wavelet subband statistics for texture retrieval. In *IEEE International Conference on Image Processing (ICIP)*, pages 933–936. IEEE, 2008.

[71] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, page 282–289, 2001.

[72] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep Laplacian pyramid networks for fast and accurate super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 624–632, 2017.

[73] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[74] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages II–104, 2004.

[75] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4681–4690, 2017.

[76] Chia-Wen Lin and Nai-Chia Cheng. Video background inpainting using dynamic texture synthesis. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1559–1562, 2010.

[77] David G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[78] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Zhiyong Gao, and Ming-Ting Sun. Deep kalman filtering network for video compression artifact reduction. In *European Conference on Computer Vision (ECCV)*, pages 568–584. Springer, 2018.

[79] Vijay Mahadevan, Weixin Li, Viral Bhalodia, and Nuno Vasconcelos. Anomaly detection in crowded scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1975–1981, 2010.

[80] Stéphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, Inc., USA, 3rd edition, 2008.

[81] Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

[82] Stéphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society*, 374(2065):20150203, 2016.

[83] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations (ICLR)*, 2016.

[84] Nikolai Matni, Alexandre Proutiere, Anders Rantzer, and Stephen Tu. From self-tuning regulators to reinforcement learning and back again. In *IEEE Conference on Decision and Control (CDC)*, pages 3724–3740, 2019.

[85] Sebastian Mika, Bernhard Schölkopf, Alex J. Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel PCA and de-noising in feature spaces. In *Advances in Neural Information Processing Systems (NIPS)*, pages 536–542, 1999.

[86] Adeel Mumtaz, Emanuele Coviello, Gert R. G. Lanckriet, and Antoni B. Chan. Clustering dynamic textures with the hierarchical em algorithm for modeling video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7):1606–1621, 2012.

[87] Adeel Mumtaz, Emanuele Coviello, Gert R. G. Lanckriet, and Antoni B. Chan. A scalable and accurate descriptor for dynamic textures using bag of system trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(4):697–712, 2015.

[88] Thanh Tuan Nguyen, Thanh Phuong Nguyen, and Frédéric Bouchara. Completed local structure patterns on three orthogonal planes for dynamic texture recognition. In *IEEE International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6, 2017.

[89] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2161–2168, 2006.

[90] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59, 1996.

[91] Peter Van Overschee and Bart De Moor. N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1):75–93, 1994.

[92] Athanasios Papoulis and S.U. Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill series in electrical engineering: Communications and signal processing. McGraw-Hill, 2002.

[93] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[94] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *European Conference on Computer Vision (ECCV)*, pages 143–156. Springer, 2010.

[95] Renaud Péteri, Sándor Fazekas, and Mark J. Huiskes. DynTex : a Comprehensive Database of Dynamic Textures. *Pattern Recognition Letters*, 2010. `http://projects.cwi.nl/dyntex/`.

[96] Xianbiao Qi, Chun-Guang Li, Guoying Zhao, Xiaopeng Hong, and Matti Pietikäinen. Dynamic texture and scene classification by transferring deep image features. *Neurocomputing*, 171:1230–1241, 2016.

[97] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7229–7238, 2018.

[98] Yuhui Quan, Yan Huang, and Hui Ji. Dynamic texture recognition via orthogonal tensor dictionary learning. In *IEEE International Conference on Computer Vision (ICCV)*, pages 73–81, 2015.

[99] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[100] Avinash Ravichandran, Rizwan Chaudhry, and René Vidal. Categorizing dynamic textures using a bag of dynamical systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):342–353, 2012.

[101] Robert P. Roesser. A discrete state-space model for linear image processing. *IEEE Transactions on Automatic Control*, 20(1):1–10, 1975.

[102] Sam T. Roweis, Lawrence K. Saul, and Geoffrey E. Hinton. Global coordination of local linear models. In *Advances in neural information processing systems*, pages 889–896, 2002.

[103] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[104] Alexander Sagel, Dominik Meyer, and Hao Shen. Texture retrieval using scattering coefficients and probability product kernels. In *International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 506–513. Springer, 2015.

[105] Payam Saisan, Gianfranco Doretto, Ying Nian Wu, and Stefano Soatto. Dynamic texture recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages II–II, 2001.

[106] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks (ICANN)*, pages 92–101. Springer, 2010.

[107] Bernhard Schölkopf. Causality for machine learning. *arXiv preprint arXiv:1911.10500*, 2019.

[108] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[109] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[110] Maximilian Seitzer. Fréchet inception distance (fid score) in pytorch. `https://github.com/mseitzer/pytorch-fid`, 2019.

[111] Uri Shalit and Gal Chechik. Coordinate-descent for learning orthogonal matrices through givens rotations. In *International Conference on Machine Learning (ICML)*, pages 548–556, 2014.

[112] Sergei L Shmakov. A universal method of solving quartic equations. *International Journal of Pure and Applied Mathematics*, 71(2):251–259, 2011.

[113] Nikolay Skarbnik. Local binary patterns. MATLAB Package, 2015.

[114] Didier Sornette. *Critical phenomena in natural sciences: chaos, fractals, self-organization and disorder: concepts and tools*. Springer Science & Business Media, 2006.

[115] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[116] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning (ICML)*, pages 843–852, 2015.

[117] Martin Szummer and Rosalind W. Picard. Temporal texture modeling. In *IEEE International Conference on Image Processing (ICIP)*, volume 3, pages 823–826, 1996.

[118] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[119] Matthew Tesfaldet, Marcus A. Brubaker, and Konstantinos G. Derpanis. Two-stream convolutional networks for dynamic texture synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6703–6712, 2018.

[120] Dmitry Ulyanov. Deep image prior: Project website. `https://dmitryulyanov.github.io/deep_image_prior`, 2008.

[121] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9446–9454, 2018.

[122] René Vidal and Avinash Ravichandran. Optical flow estimation & segmentation of multiple moving dynamic textures. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 516–521. IEEE, 2005.

[123] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems (NIPS)*, pages 613–621, 2016.

[124] Li Wang and Dong-Chen He. Texture classification using texture spectrum. *Pattern recognition*, 23(8):905–910, 1990.

[125] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2746–2754, 2015.

[126] Jianwen Xie, Ruiqi Gao, Zilong Zheng, Song-Chun Zhu, and Ying Nian Wu. Learning dynamic generator model by alternating back-propagation through time. In *AAAI Conference on Artificial Intelligence*, pages 5498–5507, 2019.

[127] Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. Synthesizing dynamic patterns by spatial-temporal generative convnet. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7093–7101, 2017.

[128] Yong Xu, Yuhui Quan, Haibin Ling, and Hui Ji. Dynamic texture classification using dynamic fractal analysis. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1219–1226, 2011.

[129] Jianchao Yang, John Wright, Thomas Huang, and Yi Ma. Image super-resolution as sparse representation of raw image patches. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[130] Lu Yuan, Fang Wen, Ce Liu, and Heung-Yeung Shum. Synthesizing dynamic texture with closed-loop linear dynamic system. In *European Conference on Computer Vision (ECCV)*, pages 603–616. Springer, 2004.

[131] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International Conference on Curves and Surfaces*, pages 711–730. Springer, 2010.

[132] Kai Zhang, Ivor W Tsang, and James T Kwok. Improved Nyström low-rank approximation and error analysis. In *International Conference on Machine Learning (ICML)*, pages 1232–1239, 2008.

[133] Guoying Zhao and Matti Pietikainen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):915–928, 2007.

[134] Xiaochao Zhao, Yaping Lin, Li Liu, Janne Heikkilä, and Wenming Zheng. Dynamic texture classification using unsupervised 3d filter learning and local binary encoding. *IEEE Transactions on Multimedia*, 21(7):1694–1708, 2019.